Chapter 33

SPRUHZ6H–October 2014–Revised November 2016

Initialization

This chapter introduces the steps of the device initialization.

Topic **Page**

33.1 Initialization Overview

This chapter provides an overview of the requirements to initialize the device from power on to operating system (OS) and application execution, the overall initialization process (including hardware- and software-related steps), the general ROM code operational requirements, and behavior expectations.

33.1.1 Terminology

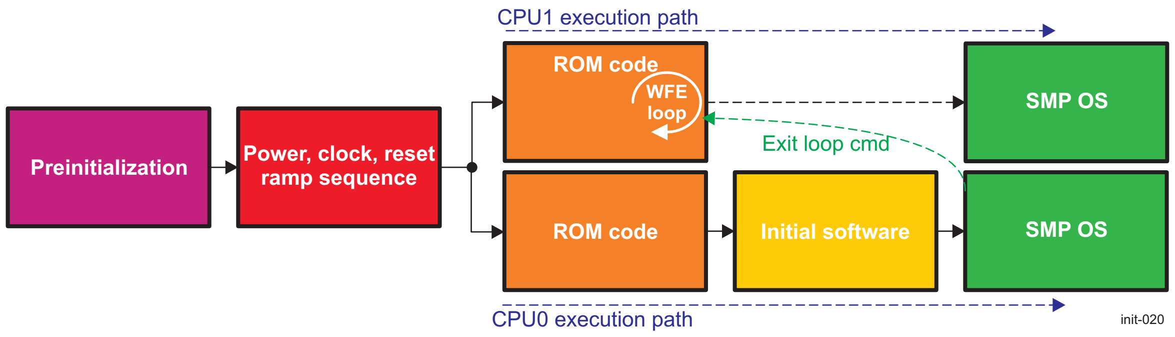
- **Bootstrap:** Initial software launched by the ROM code during the memory booting phase
- **Configuration Header (CH):** Optional structure that precedes the initial software and allows the redefinition of the ROM code default settings
- **Downloaded software:** Initial software downloaded into the internal static RAM (SRAM) by the ROM code during the peripheral booting phase
- **eFuse:** A one-time programmable memory location usually set at the factory
- **Flash loader:** Downloaded software launched by the ROM code during the preflashing stage. It also programs an image in external memories.
- **Initial software:** Software executed by any of the ROM code mechanisms (memory booting or peripheral booting). Initial software is a generic term for bootstrap and downloaded software.
- **Memory booting:** ROM code mechanism that consists of executing initial software from external memory
- **Master CPU:** The ARM® Cortex®-A15 MPCore™ CPU for which CPU-ID is 0. It configures the multicore platform and starts the ROM code to ensure device booting from a mass storage memory (memory booting) or a peripheral interface (peripheral booting).
- **Peripheral booting:** ROM code mechanism that consists of polling selected interfaces, downloading, and executing initial software (in this case, downloaded software) in the internal RAM
- **Permanent booting device:** Memory device containing, by default, the image to be executed during the booting sequence. It is the default memory booting device. The permanent booting device is used after a warm reset if no software booting configuration is programmed.
- **Preflashing:** A specific case of peripheral booting where the ROM code mechanism is used to program the external flash memory
- **ROM Code:** The on-chip software in device ROM that implements booting
- **ROM Code-controlled Boot Phase:** This phase covers the sequence operations from the time the platform releases the reset to the time first user- or customer-owned software starts execution. This phase is fully controlled by the device ROM code.
- **Save-And-Restore (SAR) RAM memory:** On-chip RAM memory that is not cleared after warm resets or wakeups from low-power modes
- **Slave CPU:** The ARM Cortex-A15 MPCore CPU for which CPU-ID is 1. It is brought to the wait-for-event (WFE) state by the ROM code, waiting to be woken up by the master CPU.
- **Software booting configuration:** A logical structure stored in the SAR memory that allows the redefinition of the ROM code default settings when booting after a warm reset

33.1.2 Initialization Process

Figure 33-1 is an overview of the initialization process and its steps:

- **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 code:** Responsible for finding, for downloading, and for executing the initial software by using the master CPU
- **Initial software:** Software that prepares and passes control to application software or to the high-level operating system (HLOS)
- **Symmetric multiprocessing (SMP)-capable HLOS** or application (primarily for diagnostics)

Figure 33-1. Initialization Process



The first two steps in the initialization process are hardware-oriented; however, they require an understanding of the process of configuring these system interface pins (balls on the device), which have software-configurable functionality. This configuration is an essential part of the chip configuration and is application-dependent. This chapter discusses these system-interface pins and the associated configuration registers that are vital to the correct initialization of the device.

33.2 Preinitialization

To accomplish a successful boot-up operation, certain hardware configuration settings must be in place. Clock, reset, and power connections, as well as pins involved in setting the boot memory space for the master CPU, must be connected and driven correctly to successfully ini

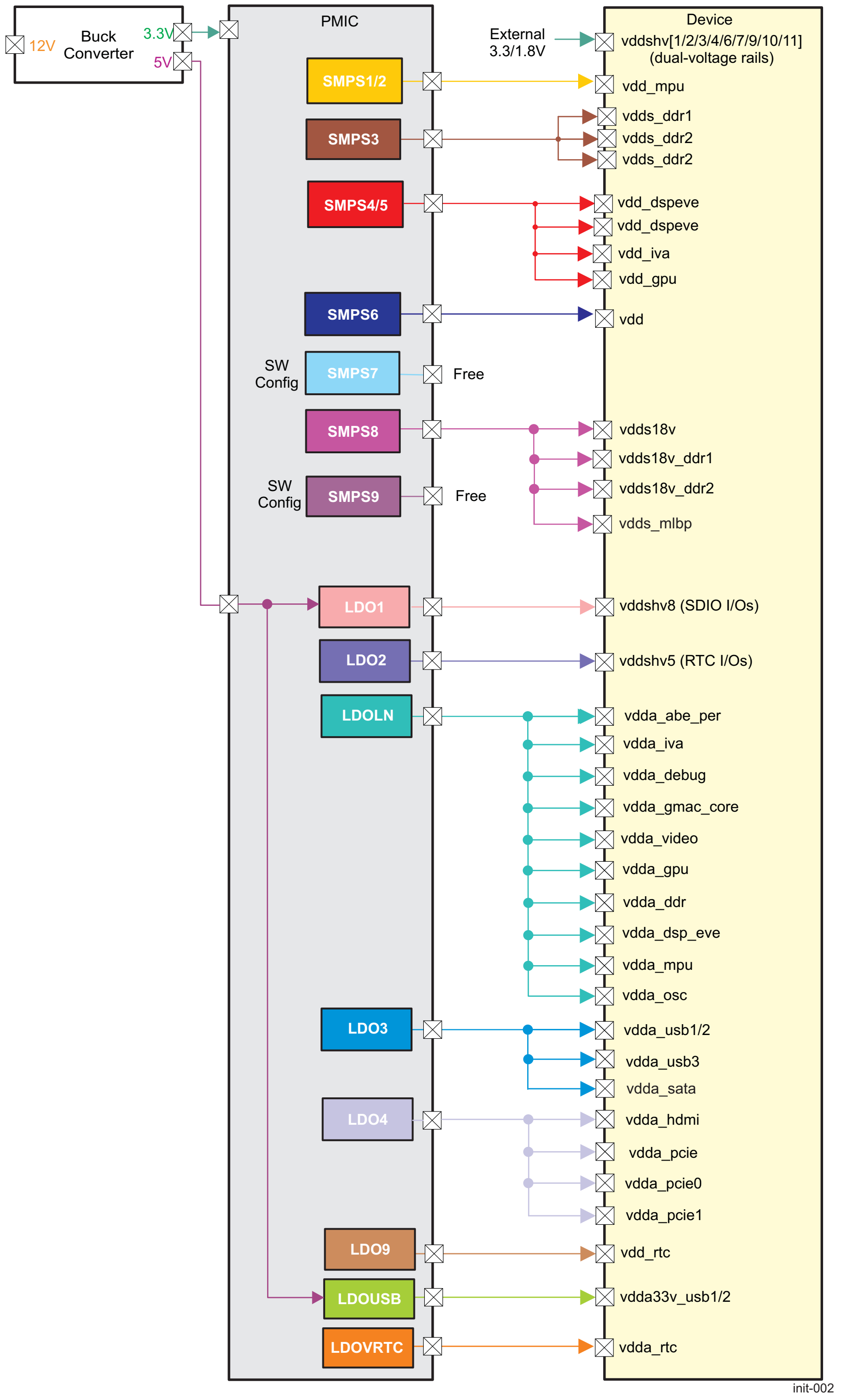
tialize the device. The following sections describe the specific requirements for the preinitialization stage.

33.2.1 Power Requirements

The device can be supplied by an external power-management integrated circuit (PMIC). TI provides a global solution with the device connected to the power-management IC companion chip. Refer to *Data Manual* for information about the power-management IC companion chips supported for this device.

Figure 33-2 shows typical power connections between the device and a PMIC companion chip.

Figure 33-2. Power Supply Connections Example



NOTE: Figure 33-2 is an example of power connections between the device and the PMIC, representing only one of the multiple PMIC/OTP options.
These connections depend on the actual application, PMIC, and OTP used. Refer to the Device *Data Manual*, respective PMIC Data Sheet, and all related application notes before starting a new design.

Table 33-1 describes the device power balls.

Table 33-1. Device Power Balls

Voltage Ball Name	Subsystems and Peripherals
vdd	Subsystems and modules supplied by CORE voltage domain
vdd_mpu	MPU voltage domain
vdd_iva	IVA voltage domain
vdd_dspeve	DSP and EVE voltage domain
vdd_gpu	GPU voltage domain
vdd_rtc	RTC voltage domain
vdds18v	1.8V I/Os
vdds_mlbp	MLBSS I/Os
vdds_ddr1	EMIF channel 1 (1.8V for DDR2 mode/ 1.5V for DDR3 mode)
vdds_ddr2	EMIF channel 2 (1.8V for DDR2 mode/ 1.5V for DDR3 mode)
vdds18v_ddr1	EMIF channel 1 bias
vdds18v_ddr2	EMIF channel 2 bias
vdda_video	Analog power supply for DPLL_VIDEO0/1
vdda_ddr	Analog power supply for DPLL_DDR and DDR HSDIVIDER
vdda_mpu	Analog power supply for DPLL_MPU
vdda_abe_per	Analog power supply for DPLL_ABE, DPLL_PER, PER HSDIVIDER
vdda_gmac_core	Analog power supply for DPLL_GMAC DPLL_CORE, and CORE HSDIVIDER
vdda_hdmi	Analog power supply for HDMI I/Os and DPLL_HDMI
vdda_pcie	Analog power supply for DPLL_PCIE_REF and APLL_PCIE
vdda_pcie0	Analog power supply for PCIE0 I/Os
vdda_pcie1	Analog power supply for PCIE1 I/Os
vdda_iva	Analog power supply for DPLL_IVA
vdda_gpu	Analog power supply for DPLL_GPU
vdda_dsp_eve	Analog power supply for DPLL_DSP and DPLL_EVE
vdda_usb3	Analog power supply for USB1 DPLL_USB_OTG_SS and USB3.0 I/Os
vdda_usb1	Analog power supply for USB1 DPLL_USB and USB2.0 I/Os (1.8 V)
vdda33v_usb1	Analog power supply for USB1 I/Os (3.3 V)
vdda_usb2	Analog power supply for USB2 I/Os (1.8 V)
vdda33v_usb2	Analog power supply for USB2 I/Os (3.3 V)
vdda_sata	Analog power supply for SATA I/Os and DPLL_SATA
vdda_debug	Analog power supply for DPLL_DEBUG
vddshv1	Dual-voltage VIN2 group I/Os
vddshv2	Dual-voltage VOUT groupI/Os
vddshv3	Dual-voltage GENERAL group I/Os
vddshv4	Dual-voltage MMC4 group I/Os
vddshv5	Dual-voltage RTC group I/Os
vddshv6	Dual-voltage VIN1 group I/Os
vddshv7	Dual-voltage WIFI group I/Os
vddshv8	Dual-voltage MMC1 group I/Os
vddshv9	Dual-voltage RGMII group I/Os
vddshv10	Dual-voltage GPMC group I/Os
vddshv11	Dual-voltage MMC2 group I/Os
vdda_osc	System HF OSC0/1 XTAL oscillators
vdda_rtc	RTC bias and LF RTC XTAL oscillator

NOTE: Table 33-1 is for informative purposes. For a complete description of the power balls of your device, please refer to the *Device Data Manual*.

For more information about power management, see *Clock Management Functional Description*, in Chapter 3, *Power, Reset, and Clock Management*.

NOTE: MLB and DDR2 memories are not supported in this family of devices.

33.2.2 Interaction With the PMIC Companion

The ROM code does not perform any I^{2}C transactions with the PMIC. The following system conditions must be met to perform device initialization:

- The USB transceivers (USB2.0), internal to device, are powered on reset, as expected by the USB peripheral booting feature.
- The SD card cage must be appropriately powered before entering the SD card boot feature on any reset.
- Devices must be appropriately powered and be up and ready at platform startup:
  - eMMC
  - QSPI
  - GPMC
  - SATA

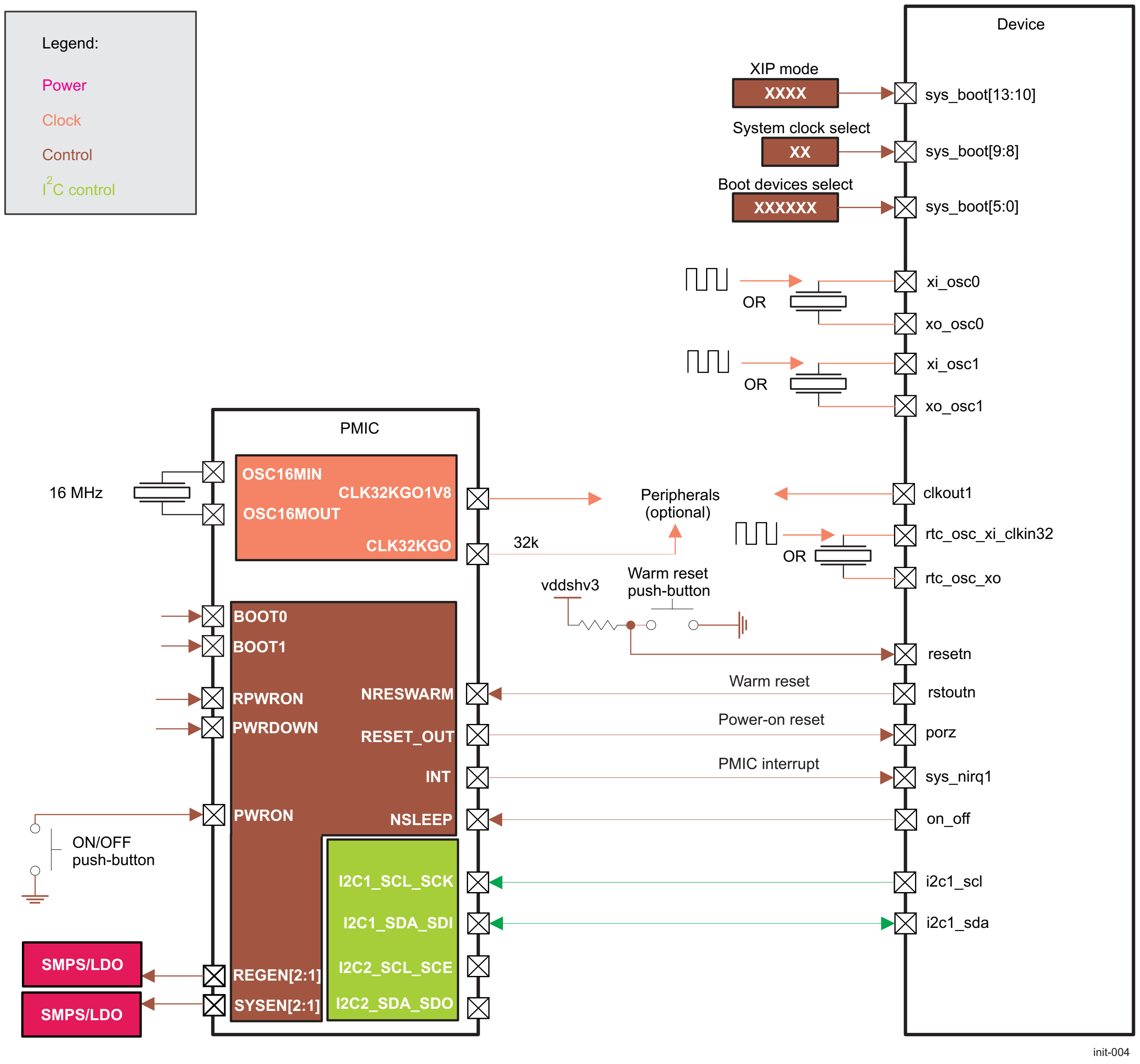
NOTE: The ROM code does not interact with the PMIC companion chip over an I^{2}C or SPI interface.

33.2.3 Clock, Reset, and Control

33.2.3.1 Overview

Figure 33-3 shows the clock and reset environment where clocks and reset-related signals are gathered at the system level, the system-expansion signals, and the crystal oscillator connection.

Figure 33-3. Clock, Reset, and Control Environment Overview



NOTE: Figure 33-3 is a typical example of clock, reset and control connections between the device and a PMIC. Refer to the Device *Data Manual* for the supported PMIC(s) for your device and for more information on these balls.
For PMIC ball description, see the respective PMIC Data Sheet.

The main features of the system interface are:

- Accepts crystals connected between device xi_osc0 and xo_osc0 pads, and xi_osc1 and xo_osc1 pads to generate SYS_CLK1 and SYS_CLK2, resepectively

- Accepts external LVCMOS clock sources connected to xi_osc0, and xi_osc1 pads for SYS_CLK1 and SYS_CLK2, respectively
- A 32-kHz LVCMOS clock input for RTC operation
- Up to four reference clock inputs
- Up to three configurable clock outputs
- sysboot[15:0] input signals to define the boot mode, system clock speed, and GPMC in XIP mode
- Two reset sources
  - Power-on reset (cold reset)
  - Warm reset
- Three external interrupt lines (sys_irq1, sys_irq2, and nmi)
- Four external DMA requests

33.2.3.2 Clocking Scheme

The device operation requires external input clocks, as follows:

- 32k clock: A 16-MHz crystal is connected to the PMIC companion chip that embeds the 32k-oscillator (a square CMOS 32-kHz clock can also be delivered on the OSC16MIN pin of Power-management IC, if the 32k-crystal is connected on another device of the system). The resulting 32k-clock is delivered to the entire system on two outputs:
  - CLK32KGO
  - CLK32KGO1V8
- System clocks: The device supports two system clocks with two clock sources each:
  - SYS_CLK1 (main system clock):
    - Crystal on OSC0 pins. An internal oscillator (OSC0) embedded in the device is used.
    - External LVCMOS clock on xi_osc0
  - SYS_CLK2 (optional system clock):
    - Crystal on OSC1 pins. An internal oscillator (OSC1) embedded in the device is used.
    - External LVCMOS clock on xi_osc1
- The device can deliver digital clocks to peripherals ICs.

The device provides wide choice of clocks that can be delivered on clkout[1:3] pads to companion devices. For more information, see Section 3.3, *PRCM Subsystem Environment*, in Chapter 3, *Power, Reset, and Clock Management*. For more information on pad multiplexing, see Chapter 18, *Control Module*.

Table 33-2 lists the mapping for the device clock input sources. Table 33-3 lists the PMIC clock requirements.

Table 33-2. Mapping for Input Sources

Clock	Clock Source	Ball Mapping	Frequency Range/List	Type
SYS_CLK1	Internal oscillator 0 (OSC0)	xi_osc0 and xo_osc0	19.2, 20, and 27 MHz	Crystal connection pins
	External	xi_osc0	19.2, 20, and 27 MHz	External LVCMOS
SYS_CLK2	Internal oscillator 1 (OSC1)	xi_osc1 and xo_osc1	19.2 ÷ 32 MHz	Crystal connection pins
	External	xi_osc1	12 ÷ 38.4 MHz	External LVCMOS

Table 33-3. PMIC Clock Requirements[1]

Clock Source	Mapping	Frequency Range/List	Type
Internal oscillator	OSC16MIN and OSC16MOUT	16.384 MHz	Crystal connection pins
External	OSC16MIN	32.768 kHz (nom.)	External LVCMOS

(1) The PMIC does not need any external clock to operate properly. PMIC can run on internal RC oscillator. The 16-MHz oscillator is in place to provide an accurate 32-kHz clock to its internal RTC, to the SoC or device in the system. See the corresponding PMIC datasheet for details.

CAUTION

Clock configurations depend on core voltage, and maximum clock frequencies may not apply to production. For more information, see the *Device Data Manual*.

33.2.3.3 Reset Configuration

33.2.3.3.1 ON/OFF Interconnect and Power-On-Reset

The entire system is typically awakened by an ON/OFF push button connected to the PMIC chip. This signal belongs to the VSYS - system power domain and is active low (the PMIC internal pullup ties it to VSYS). The PMIC power-up event is propagated through its NRESPWRON output pin to the device porz pad (that is, the PRM SYS_PWRON_RST signal) when the PMIC power-up sequence is achieved. The device porz input pin is held low all the time during VDD core and I/O power-up.

33.2.3.3.2 Warm Reset

A warm reset can be asserted by the device, by an external button (typically for a development platform), or by any other chip connected to it (normally tied to PMIC companion NRESWARM output pin).

The device warm reset pad (resetn - device signal SYS_WARM_IN_RST) is used to trigger a warm reset on the device, which resets part of the device when it has already booted (for example, to recover from a software crash). When an internal device reset occurs, SYS_NRES_WARM_OUT output and device pad rstoutn go low and reset all the peripherals.

The device releases the SYS_NRES_WARM_OUT output signal after SYS_PWRON_RST is deasserted.

33.2.3.3.3 Peripheral Reset by GPIO

Most peripherals can be reset and powered on or off by GPIO. By default, under POR, most device signals are in safe mode with a default value driven by the I/O cell. The value is driven by an internal pullup or pulldown. Depending on the peripheral reset active level, users must select one GPIO or another (according to the reset value).

Once POR is released, the value on the pad is driven by the default configuration of the device control module. Most of time, this configuration is aligned with the default value selected on the I/O cell.

The next step is application-dependent: Users must configure the device registers to validate GPIO use and the default configuration of the control module.

33.2.3.3.4 Warm Reset Impact on GPIOs

When a warm reset event occurs:

- The GPIO controller is reset. Consequently, the GPIO is automatically turned in input mode.
- The control module is not reset. Information related to signal multiplexing mode and pullup or pulldown configuration is still valid.

Therefore, when a warm reset event occurs, the output buffer is disabled. Consequently, two different behaviors can be defined with regard to what is expected by the platform:

- GPIO sensitive to warm reset:

  To prevent a floating pad, user software is designed to have the internal pad PU and PD resistors enabled immediately, before the software warm reset action, because the warm reset-sensitive GPIO controllers will change I/O direction to input after an device warm reset. This is necessary if the warm reset-sensitive GPIO controller pin has been configured for output before the warm reset occurrence. The pulls-enabled-before-warm-reset condition should be set by default in case the user has configured a GPIO as an input, because in this case the user is expected to have enabled the internal PU and PD pads during GPIO configuration (unless external pull resistors were used).

  NOTE: If the PU and PD resistors are enabled immediately by software after a POR (cold reset) for a GPIO that is planned to be used only as an output, then unnecessary power consumption can occur.

  While the dynamically-enable-the-pull-just-before-warm-reset condition is possible during a software warm reset (because the user software is aware of the exact moment a warm reset event occurs), it is not possible when the warm reset is triggered by hardware (for example, a watchdog reset, SYS_NRESWARM signal assertion, and so forth), because the software is not aware of the exact moment of these warm reset assertions.
- GPIO not sensitive to warm reset:

  To avoid getting a floating signal during (and after) a warm reset event and to keep the same value that was driven before the reset, users must align the pull value with the drive value each time a dedicated GPIO register is accessed.

  NOTE: To avoid unnecessary consumption, the user software must ensure that internal pull resistor is disabled when the GPIO buffer is driving.

For the description of the reset sequences and information about the device reset management, see Section 3.5, *Reset Management Functional Description*, in Chapter 3, *Power, Reset and Clock Management*. For more information about the device reset sequences, see the device *Data Manual*.

33.2.3.4 PMIC Control

- I^{2}C:

  The device interfaces: system interface I/Os, and I2C1 are involved in system interactions between the device and external power, reset and clock management IC companions.

  The device and PMIC companion implement the basic power-management interface, as follows:

  – 32-kHz clock single input

  – Two system resets: power-on (cold) reset and warm reset

  – One interrupt
- INT:

  PMIC companion device can activate its output interrupt request signal (INT) at any time when requires the device to monitor its activity. When receiving such an interruption, the device checks, through the I^{2}C, to determine the source of the interrupt. INT pin is active low.

33.2.3.5 PMIC Request Signals

The PMIC drives three external enable-output signals, which allows switching on some external resources at different stages of the power-up sequence:

- REGEN1 and REGEN2 are driven high at the beginning of the power-up sequence, before any internal power source is turned on. They belong to the VSYS power domain. REGEN1 can typically be used for buck boost control.
- SYSEN is driven high immediately after the VCORE power output is turned on. SYSEN belongs to the VIO power domain.

The PMIC companion chip can receive two power resource requests: ENABLE1 and NSLEEP. These pins allow an external device to request PMIC internal resources. The PMIC companion power behavior when pins are activated must be programmed after the first boot (resources and ENABLE pins allocated by group, resource behavior upon group activation, and so forth).

The use of NSLEEP is especially required when PMIC chip is in sleep mode, because it cannot handle an I^{2}C command. Any device that requires the PMIC companion resource to wake up must first activate its associated ENABLE signal. All power-management chip power regulators are off or in sleep mode when it is in sleep mode.

33.2.4 Sysboot Configuration

The device implements 16 sampled-on-reset sysboot pads.

Table 33-4. Sysboot Pads Description[1]

Pad	Description
sysboot[15]	Must be pulled to vdd for proper device operation (SR1.1). Used to permanently disable the internal PU/PD resistors on pads gpmc_a[27:24, 22:19] (SR2.0).
sysboot[14]	Must be pulled to vss for proper device operation.
sysboot[13:10]	Used to configure the GPMC interface when booting from XIP/NAND memory connected to GPMC.
sysboot[9:8]	Selects the SYS_CLK1 clock speed. Must be set correctly according to the speed of the connected crystal.
sysboot[7:6]	Sector offset for the location of the redundant SBL images in QSPI.
sysboot[5:0]	Select interfaces or devices for the booting list

[1] Boards should be implemented with a mechanism to easily modify the pull-up/down state to enable any future modifications.

Table 33-5. MMC2 Configuration (SR2.0)[1]

sysboot[15]	mmc2_dat[7:0] Pull-down resistors
0b0	Software re-configuration of pull resistors is allowed. Internal pull-downs on gpmc_a[n:19] are enabled by default to allow GPMC boot. Pulling low gpmc_a[n:19] is required in order to access the low-order address locations in the flash memory during boot (n = [27:24, 22:19] and depends on the memory volume).
0b1	Internal pull-down resistors permanently disabled to avoid contention with the recommended per eMMC standard pull-ups that should be present on PCB. Software re-configuration of internal pull resistors is disabled.

[1] When internal pull-downs are permanently disabled (sysboot[15] = 1), gpmc_a[n:19] pads (where n = [27:24, 22:19]) require external pull-downs to enable GPMC boot.

All sysboot pads are sampled and latched onto the CTRL_BOOTSTRAP register (in control module) after POR. After booting, these pads can be used for other functions such as GPIOs, and the associated register bit field is not updated by the new functionality. For more information about pad multiplexing configuration, see Section 18.4.6.1.1, *Pad Configuration Registers*, in Chapter 18, *Control Module*.

NOTE: If used as GPIOs, the sysboot[15:0] pads must be used only in output mode to ensure that the input values always match a certain hardware predefined boot pattern, interpreted after each POR.

33.2.4.1 GPMC Configuration for XIP/NAND

Table 33-6 describes the GPMC interface configuration used in XIP, fast XIP and NAND modes controlled by sysboot[13:10].

Table 33-6. GPMC for XIP/NAND Configuration

sysboot[13]	Bus Width
0b0	8-bit
0b1	16-bit
sysboot[12:11]	**A/D-muxed/non-muxed Device on CS0**
0b00	Non-muxed device
0b01	A/D-muxed device
sysboot[10]	**Wait-pin Monitoring for Read Accesses**
0b0	Disabled
0b1	Enabled

33.2.4.2 System Clock Speed Selection

There are three crystal speeds available to be selected by means of sysboot[9:8], as described in Table 33-7. User is responsible to set the correct value depending on the actual clock supplied to the device.

Table 33-7. System Clock (SYS_CLK1) Speed Selection

sysboot[9:8]	SYS_CLK1 Speed
0b00	Reserved
0b01	20 MHz
0b10	27 MHz
0b11	19.2 MHz

33.2.4.3 QSPI Redundant SBL Images Offset

Four options are available to set the offset between the redundant SBL images as described in Table 33-8. If not using the redundant SBL feature, there is no change required to sysboot pins as only the primary image is used and the sector offset is a don't care.

Table 33-8. Offset Between Redundant Images

sysboot[7:6]	Offset
0b00	64 KiB
0b01	128 KiB
0b10	256 KiB
0b11	512 KiB

33.2.4.4 Booting Device Order Selection

The ROM code creates the device list (order) based on information gathered from these locations:

- The first location is the sysboot[5:0] external configuration pins sensed in the device CTRL_BOOTSTRAP register. The sysboot[5:0] configuration pads have two main purposes: configure ROM code software in terms of interfaces and devices used for booting and configuring hardware after a POR or cold reset.
- The second location is the software booting configuration stored in the nonvolatile SAR memory, and used after a global warm reset.

The SYSBOOT pins are used to index a booting device list from a table with possible booting scenarios. The order of examined booting devices is from the first to the third devices.

The following names are used in the tables:

- Memory types:

- Execute in place (XIP): NOR (CFI) flash memory or other XIP device
- NAND: NAND flash memories (non-XIP)
- SD: Removable SD card device
- eMMC: eMMC™ memory device
- QSPI_1: 1-bit SPI flash memories
- QSPI_4: 4-bit (Quad) SPI flash memories
- SATA: SATA-compatible devices such as solid state drives (SSDs) or hard-disk drives (HDDs)

- Peripheral interfaces:
  - USB: HS USB 2.0 interface
  - UART: UART interface
- Table 33-9 lists the permanent booting devices in bold typeface.

The above boot modes are described in detail in Section 33.3.5, *Peripheral Booting* and in Section 33.3.7, *Memory Booting*.

NOTE: If no valid software booting configuration is found after a warm reset, the ROM code builds a device list featuring only the permanent booting devices (in **bold**)

NOTE: Users can force the execution of the peripheral booting procedure after warm reset by using the software booting configuration feature.

Table 33-9 lists the booting device order selected by ROM code depending on sysboot[5:0] pins.

Table 33-9. Booting Devices Order

sysboot[5:4]	sysboot[3:0]	First Device	Second Device	Third Device
		Peripheral Preferred Booting		
0b00	0b0000	USB	**eMMC**	
0b00	0b0001	USB	**NAND**	
0b00	0b0010	USB	**SD**	**eMMC**
0b00	0b0011	USB	**SATA**	**SD**
0b00	0b0100	USB	UART	**XIP**
0b00	0b0101	SD	**XIP**	
0b00	0b0110	SD	**QSPI_1**	
0b00	0b0111	SD	**QSPI_4**	
0b00	0b1010	SD	**Fast XIP**	
		Recovery/Upgrade or Development Booting		
0b01	0b0000	**USB**		
0b01	0b0011	**UART**		
0b01	0b0100	**SD**	USB	
0b01	0b0101	**SD**	USB	
0b01	0b0110	**SD**	USB	
0b01	0b0111	**SD**	USB	
0b01	0b1000	**SD**	USB	
0b01	0b1001	**SD**	USB	
0b01	0b1010	**SD**	USB	
0b01	0b1011	**SD**	USB	
		Memory Preferred Booting		
0b10	0b0000	**eMMC**	USB	
0b10	0b0001	**NAND**	USB	

Table 33-9. Booting Devices Order (continued)

sysboot[5:4]	sysboot[3:0]	First Device	Second Device	Third Device
0b10	0b0010	**SD**	**eMMC**	USB
0b10	0b0011	**SATA**	**SD**	USB
0b10	0b0100	**XIP**	USB	UART
0b10	0b0101	**XIP**	SD	USB
0b10	0b0110	**QSPI_1**	SD	USB
0b10	0b0111	**QSPI_4**	SD	USB
		Production Booting[1]		
0b11	0b0000	**SD**		
0b11	0b0100	**SATA**		
0b11	0b0101	**XIP**		
0b11	0b0110	**QSPI_1**		
0b11	0b0111	**QSPI_4**		
0b11	0b1000	**eMMC**		
0b11	0b1001	**NAND**		
0b11	0b1010	**Fast XIP**		
0b11	0b1011	**eMMC (boot part.)**[2]		

(1) After 10 failed loops through the device list, ROM code initiates immediate global warm reset.

(2) The ROM code boots from eMMC boot partition. If the ROM code fails to retrieve the booting image, it does not try boot from the user area contrary to other eMMC options.

33.2.4.5 Boot Peripheral Pin Multiplexing

Table 33-10 lists the code pin multiplexing configuration done by ROM code according to selected boot peripheral. These settings are not restored to default values at ROM Code exit.

NOTE: The ROM code examines the interfaces that are selected to be searched until a valid bootable interface or device is found. The activities on the pads of the searched interfaces must be considered if they are connected to any other peripherals for any other purposes (for example, a LED connected to a GPMC pad muxed internally to a GPIO).

Table 33-10. Pin Multiplexing According to Boot Peripheral

Boot Device	Boot Interface	Pads	MuxMode	Signals
eMMC	MMC2	gpmc_a[19:27], gpmc_cs[1]	MuxMode=0x1	mmc2_dat[4:7], mmc2_clk, mmc2_dat[0:3], mmc2_cmd
SD	MMC1	mmc1_clk, mmc1_cmd, mmc1_dat[0:3]	MuxMode=0x0	mmc1_clk, mmc1_cmd, mmc1_dat[0:3]
NAND	GPMC	GPMC on CS0	MuxMode=0x0	GPMC on CS0
XIP	GPMC	GPMC on CS0	MuxMode=0x0	GPMC on CS0, wait signal monitoring according to the SYSBOOT[10] setting
SATA	SATA	sata1_txp0, sata1_txn0, sata1_rxp0, sata1_rxn0	-	sata1_txp0, sata1_txn0, sata1_rxp0, sata1_rxn0
QSPI_1/QSPI_4	QSPI1	gpmc_a[13:18], gpmc_cs[2]	MuxMode=0x1	qspi1_rtclk, qspi1_d[3:0], qspi1_sclk, qspi1_cs[0]
USB	USB1	usb1_dp and usb1_dm	-	usb1_dp and usb1_dm

UART	UART3	uart2_rtsn
		uart2_ctsn
			MuxMode=0x1
			MuxMode=0x2
				uart3_txd
				uart3_rxd

33.3 Device Initialization by ROM Code

This section describes high-level booting concepts and provides basic knowledge for booting on the device.

33.3.1 Booting Overview

33.3.1.1 Booting Types

Booting is the process of starting a bootstrap from one of the booting devices.

The ROM code has two functions for booting: Peripheral booting and memory booting.

- In peripheral booting, the ROM code polls a selected communication interface such as UART or USB, downloads the executable code over the interface, and executes it in internal RAM. Downloaded software from an external host can be used to program flash memories connected to the device. This special case of peripheral booting is called preflashing; software downloaded for preflashing is called the flash loader. The flash loader burns a new client application image in external flash memory. Initial software is a generic term for bootstrap, downloaded software, and flash loader. A software (warm) reset can be performed after the image is burned.
- In memory booting, the ROM code finds the bootstrap in permanent memories such as flash memory, memory cards, or SATA SSD or HDD memory devices and executes it. This process is normally performed after a cold or warm device reset.

The ROM code detects whether the device should download software from a peripheral interface (USB or UART) by using the sysboot pad configuration. This mechanism encompasses initial flashing in production (external memory is empty) and reflashing in service (external memory is already programmed).

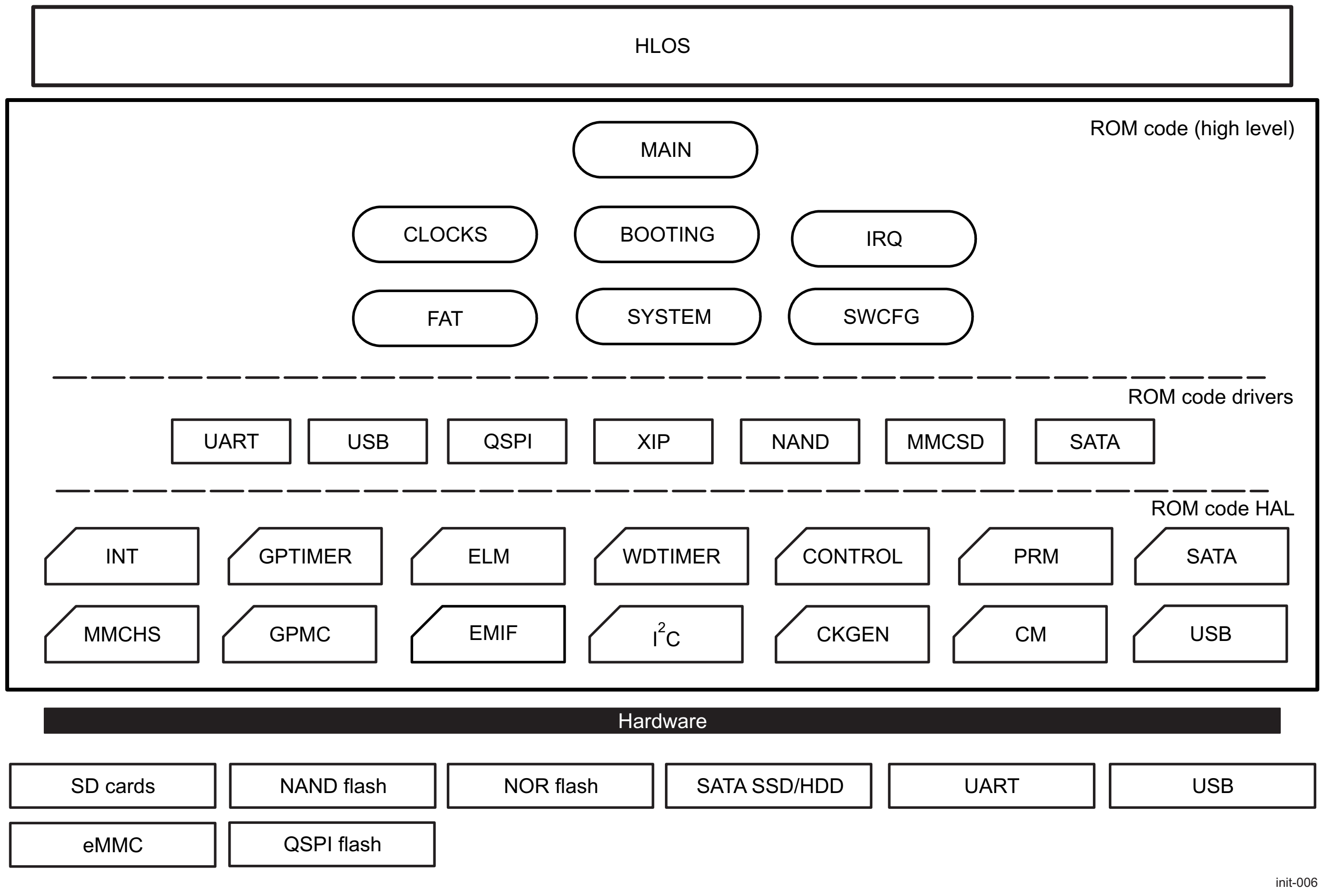
33.3.1.2 ROM Code Architecture

Figure 33-4 shows the ROM code architecture. It is split into three main layers with a top-down approach: high-level, drivers, and hardware abstraction layer (HAL). One layer communicates with a lower-level layer through a unified interface.

- The high-level layer performs the main tasks of the public ROM code: multicore startup, watchdog and clock configurations, interrupt management, and main booting routine.
- The driver layer implements the logical and communication protocols for any booting device in accordance with the interface specification.
- The HAL implements the lowest level code for interacting with the hardware infrastructure modules. End booting devices (typically external flash components) are attached to the device I/O pads.

Figure 33-4 shows the three layers with their modules.

Figure 33-4. ROM Code Architecture

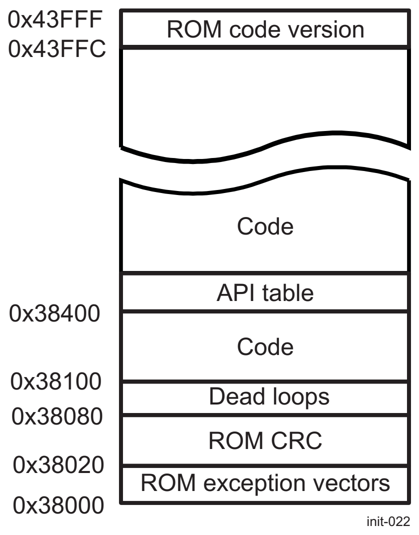


33.3.2 Memory Maps

33.3.2.1 ROM Memory Map

Figure 33-5 shows the 48-KiB ROM memory map.

Figure 33-5. ROM Memory Map



- ROM exception vectors

  Exceptions are redirected to ROM exception vectors (see Table 33-11). The reset exception is redirected to the public ROM code startup. Other exceptions are redirected to RAM handlers by loading appropriate addresses to the PC register.

Table 33-11. ROM Exception Vectors

Address	Exception	Content
0x38000	Reset	Branch to the ROM code startup
0x38004	Undefined	PC = 0x4037 F004
0x38008	Software interrupt (SWI)	PC = 0x4037 F008
0x3800C	Prefetch abort	PC = 0x4037 F00C
0x38010	Data abort	PC = 0x4037 F010
0x38014	Unused	PC = 0x4037 F014
0x38018	IRQ	PC = 0x4037 F018
0x3801C	FIQ	PC = 0x4037 F01C

- ROM code cyclic redundancy check (CRC)

  The ROM code CRC is calculated as 32-bit CRC code (CRC-32-IEEE 802.3) for the address range 0x38000–0x43FFF. The 4-byte CRC code is stored at location 0x38020.

- Dead loops

  Dead loops are branch instructions coded in ARM mode. They have multiple purposes (see Table 33-12).

Table 33-12. Dead Loops

Address	Purpose
0x38080	Undefined exception default handler
0x38084	SWI exception default handler
0x38088	Prefetch abort exception default handler
0x3808C	Data abort exception default handler
0x38090	Unused exception default handler
0x38094	IRQ exception default handler
0x38098	FIQ exception default handler

- Code

  This space is used to hold code and constant data.
- API table

  The purpose of this table is to allow external code access to system maintenance, utility, and device driver functions which are used for ensuring the ROM code boot functionality. These functions can be reused at run time by calling a fixed address hardcoded in this table.
- ROM code version

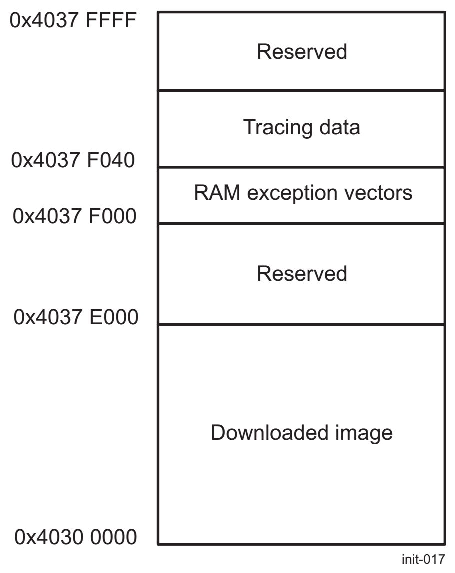
  The ROM code version consists of two BCD numbers: major and minor. It can be used to identify the ROM code release version burned in a given IC. The ROM code version is a 32-bit hexadecimal value at address 0x43FFC.

  The ROM code version numbers are:
  – 0x2601 for SR1.0
  – 0x2602 for SR1.1 and SR2.0

33.3.2.2 RAM Memory Map

The partitioning of the on-chip SRAM (L3 OCM RAM) shown in Figure 33-6 is used during the booting process. Tracing areas can also be accessed when calling API functions.

Figure 33-6. RAM Memory Map



- Downloaded image

  This space is used by the public ROM code to store a downloaded booting image. It can be up to 504 KiB.
- RAM exception vectors

  The RAM exception vectors provide an easy way to redirect exceptions to the custom handler.

Table 33-13 lists the contents of the RAM space reserved for RAM vectors. The first eight addresses are ARM instructions that load the value in the subsequent eight addresses into the PC. These instructions are executed when an exception occurs because they are called from ROM exception vectors. Undefined, SWI, unused, and FIQ exceptions are redirected to a hardcoded dead loop. Prefetch abort, data abort, and IRQ exception are redirected to predefined ROM handlers. Users can redirect an exception to another handler by writing its address to the appropriate location from 0x4037 F024 to 0x4037 F03C, or by overriding the branch (load into PC) instruction between addresses from 0x4037 F004 to 0x4037 F01C.

Table 33-13. RAM Exception Vectors

Address	Exception	Content
0x4037 F000	Reserved	Reserved
0x4037 F004	Undefined	PC = [0x4037 F024]
0x4037 F008	SWI	PC = [0x4037 F028]
0x4037 F00C	Prefetch abort	PC = [0x4037 F02C]
0x4037 F010	Data abort	PC = [0x4037 F030]
0x4037 F014	Unused	PC = [0x4037 F034]
0x4037 F018	Interrupt request (IRQ)	PC = [0x4037 F038]
0x4037 F01C	Fast interrupt request (FIQ)	PC = [0x4037 F03C]
0x4037 F020	Reserved	0x38090
0x4037 F024	Undefined	0x38080
0x4037 F028	SWI	0x38084
0x4037 F02C	Prefetch abort	Address of default prefetch abort handler(1)
0x4037 F030	Data abort	Address of default data abort handler (1)
0x4037 F034	Unused	0x38090
0x4037 F038	IRQ	Address of default IRQ handler
0x4037 F03C	FIQ	0x38098

(1) The default handlers for prefetch and data abort perform reads from CP15 debug registers to retrieve the reason for the abort:

- In case of prefetch abort: the IFAR register is read from CP15 and stored into R0. The IFSR register is read and stored into the R1 register. Then the ROM code jumps to the prefetch abort dead loop (38088h).
- In case of data abort: the DFAR register is read from CP15 and stored into R0. The DFSR register is read and stored into the R1 register. Then the ROM code jumps to the data abort dead loop (3808Ch).

- Tracing data

  This area contains trace vectors reflecting the execution path of the ROM code. Table 33-14 describes the public ROM code tracing data. For more information about ROM code tracing, see Section 33.3.9, *Tracing*.

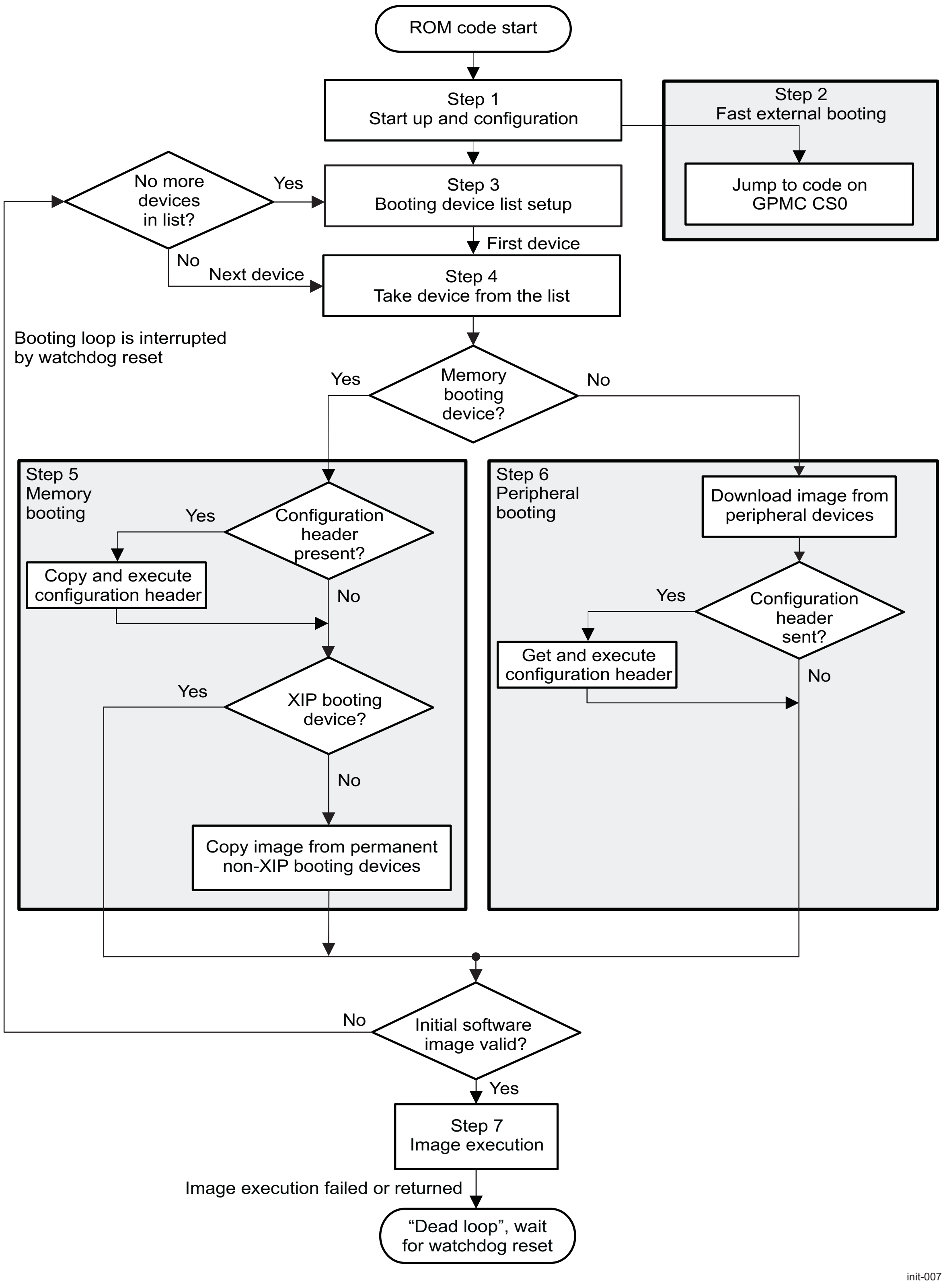
Table 33-14. Tracing Data

Address	Size (Bytes)	Description
0x4037 F040	4	Current tracing vector, word 1
0x4037 F044	4	Current tracing vector, word 2
0x4037 F048	4	Current tracing vector, word 3
0x4037 F04C	4	Current tracing vector, word 4
0x4037 F050	4	Cold reset run tracing vector, word 1
0x4037 F054	4	Cold reset run tracing vector, word 2
0x4037 F058	4	Cold reset run tracing vector, word 3
0x4037 F05C	4	Cold reset run tracing vector, word 4
0x4037 F060	4	Current copy of the PRM_RSTST register (reset reasons)

33.3.3 Overall Booting Sequence

Figure 33-7 shows the ROM code flow chart.

Figure 33-7. Overall Booting Sequence


ROM code start
Step 1
Start up and configuration
Step 2
Fast external booting
Jump to code on GPMC CS0
No more devices in list?
Yes
Step 3
Booting device list setup
First device
No
Next device
Step 4
Take device from the list
Booting loop is interrupted by watchdog reset
Yes
Memory booting device?
No
Step 5
Memory booting
Yes
Configuration header present?
Copy and execute configuration header
No
Yes
XIP booting device?
No
Copy image from permanent non-XIP booting devices
Step 6
Peripheral booting
Download image from peripheral devices
Yes
Configuration header sent?
Get and execute configuration header
No
No
Initial software image valid?
Yes
Step 7
Image execution
Image execution failed or returned
"Dead loop", wait for watchdog reset
init-007

The main loop of the booting module goes through the booting device list and tries to get an image from the currently selected booting device. The ROM code performs the following steps:

Step 1. Basic configuration and initialization. Reading of SYSBOOT pins or software booting configuration.

Step 2. The path named fast external boot is a special low-latency boot mode. It consists of a blind jump to an external addressable memory. See Section 33.3.6, *Fast External Booting.*

Step 3. A booting device list is created (see Section 33.3.4.5, *Booting Device List Setup*). The list consists of all devices to be searched for a booting image. The list is created based on the SYSBOOT pins, or the software booting configuration described in Section 33.3.4.6, *Software Booting Configuration*. The software booting configuration structure is in the SAR memory and can be written by software before executing a software reset. After a software reset, the software booting configuration has priority over the SYSBOOT configuration.

Step 4. The main loop of the booting procedure goes through the booting device list and tries to search for an image from the currently selected booting device. This loop is exited if a valid booting image is found and successfully executed or when the watchdog expires. If an image is found, ROM code executes memory booting or peripheral booting, depending on the type of the current booting device:

- Memory booting is executed when the booting device is XIP memory, NAND, QSPI, eMMC or SD, SATA SSD/HDD.
- Peripheral booting is executed when the booting device is UART or USB.

Step 5. Memory booting reads data from memory-type devices. Memory booting is described in detail in Section 33.3.7, *Memory Booting*.

Step 6. Peripheral booting downloads data from communication interfaces. Peripheral booting is described in Section 33.3.5, *Peripheral Booting*.

Step 7. The image automatically starts.

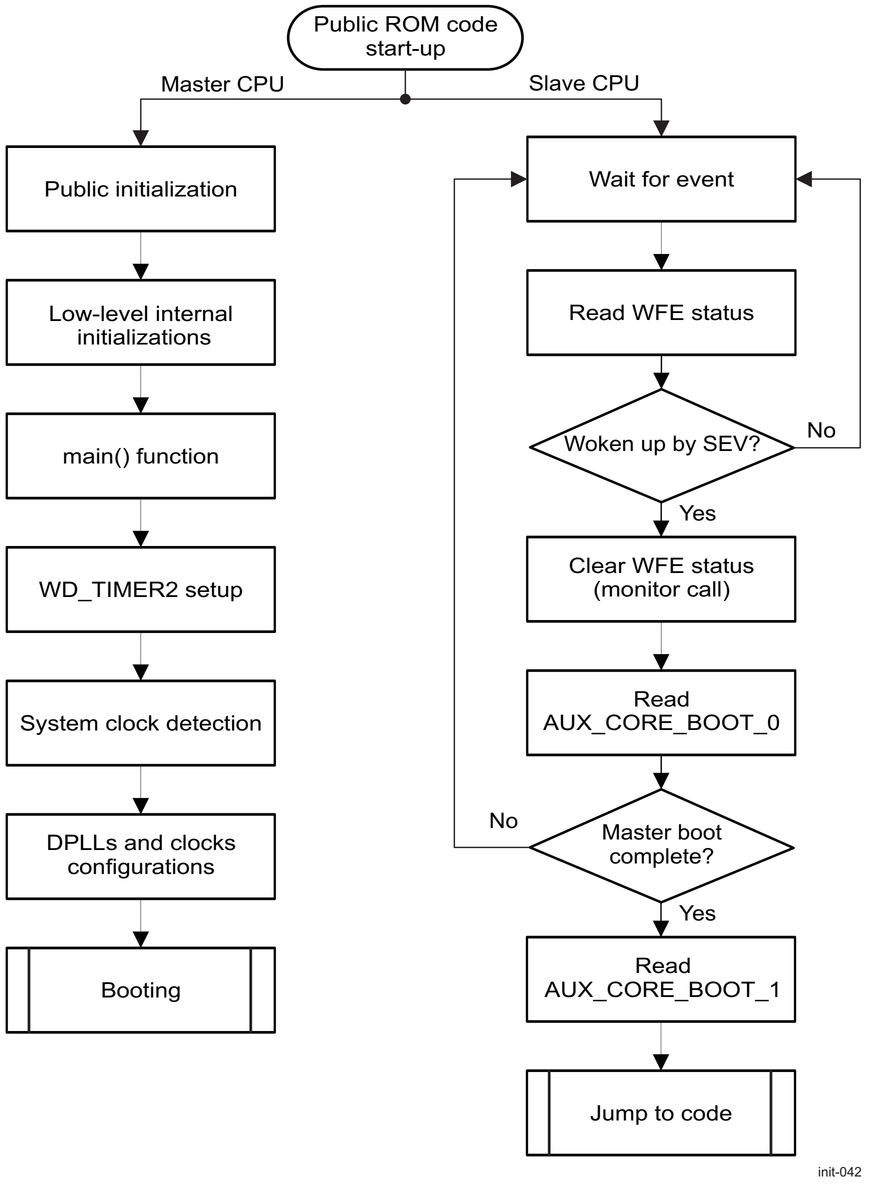
An additional feature of the booting module is the execution of the Configuration Header (CH). The CH configures the system for faster and more flexible booting from the selected permanent or peripheral booting device. The CH, which is optional, is described in *Section 33.3.8.2, Configuration Header*.

33.3.4 Startup and Configuration

33.3.4.1 Startup

Figure 33-8 shows the ROM code start-up sequence.

Figure 33-8. ROM Code Multiprocessor Start-Up Sequence



The master CPU L1 instruction cache and branch prediction mechanisms are activated as part of the public boot process. The base address of the public vector is configured to the reset vector of ROM code (0x38000). The memory management unit (MMU) remains switched off during boot (thus, L1 data cache is off). The master CPU performs the basic initialization of the public side. Next, the MMU configures WD_TIMER2 (set to 3 minutes), detects system clock, and configures the system clock. Finally, the MMU jumps to the booting routine.

No specific configuration is performed for the slave CPU, which keeps its default configuration after reset (L1 instruction and data caches off, branch prediction off, MMU off, no remap of the base address of the public vector). The slave CPU is rapidly held in wait-for-event (WFE) state. It stays in this state while the master CPU completes the public boot process and until jumping to the external software (for example, HLOS). At this stage, the external software can wake up the slave CPU by executing an SEV command.

Two internal memory-mapped registers are available to the OS for communicating start-up information. The AUX_CORE_BOOT_0 and AUX_CORE_BOOT_1 registers are in the MPU WakeupGen domain.

- AUX_CORE_BOOT_0 is used as a status register to signal the slave CPU that it must wake up after the send event operation initiated by the master CPU.
- AUX_CORE_BOOT_1 contains the physical address location to which the slave CPU must jump after wakeup.

See the memory mapping of these registers in Section 4.4.10, *MPU_WUGEN Registers* in *Dual Cortex-A15 MPU Subsystem*.

33.3.4.2 Control Module Configuration

Table 33-15 lists the Control Module registers modified at each ROM code startup. It is a Control Module requirement to be met prior to modify any of the pad control registers. These registers are not reverted back to default values (that is, to LOCK state) when ROM code completes.

Table 33-15. Control Module Registers Modified by ROM Code at Each Startup

Register	Value	Meaning
CTRL_CORE_MMR_LOCK_1	0x2FF1AC2B	Unlock Control Module registers starting at address offset 0x0000 0100 and ending at 0x0000 079F
CTRL_CORE_MMR_LOCK_2	0xF757FDC0	Unlock Control Module registers starting at address offset 0x0000 07A0 and ending at 0x0000 0D9F
CTRL_CORE_MMR_LOCK_3	0xE2BC3A6D	Unlock Control Module registers starting at address offset 0x0000 0DA0 and ending at 0x0000 0FFF
CTRL_CORE_MMR_LOCK_4	0x1EBF131D	Unlock Control Module registers starting at address offset 0x0000 1000 and ending at 0x0000 13FF
CTRL_CORE_MMR_LOCK_5	0x6F361E05	Unlock Control Module registers starting at address offset 0x0000 1400 and ending at 0x0000 1FFF

Once the booting device list is completed, the ROM code applies the pin multiplexing settings as described in Section 33.2.4.5, *Pin Multiplexing According to Boot Peripheral*.

33.3.4.3 PRCM Module Mode Configuration

Table 33-16 lists the PRCM module mode, clock control, and power control registers modified at each ROM code startup. These registers are not reverted back to default values when ROM code completes.

Table 33-16. PRCM Module Mode Registers Modified by ROM Code

Register	Field	Value
CM_L3INSTR_L3_INSTR_CLKCTRL	MODULEMODE	DISABLED
CM_L3INSTR_OCP_WP_NOC_CLKCTRL	MODULEMODE	DISABLED
CM_CM_CORE_PROFILING_CLKCTRL	MODULEMODE	DISABLED
CM_PRM_PROFILING_CLKCTRL	MODULEMODE	DISABLED
CM_CM_CORE_AON_PROFILING_CLKCTRL	MODULEMODE	DISABLED
CM_EMU_CLKSTCTRL	CLKTRCTRL	HW_AUTO
CM_DSP1_CLKSTCTRL	CLKTRCTRL	HW_AUTO
CM_DSP1_DSP1_CLKCTRL	MODULEMODE	DISABLED
PM_DSP1_PWRSTCTRL	POWERSTATE	OFF
CM_DSP2_CLKSTCTRL	CLKTRCTRL	HW_AUTO
CM_DSP2_DSP2_CLKCTRL	MODULEMODE	DISABLED
PM_DSP2_PWRSTCTRL	POWERSTATE	OFF
CM_IVA_CLKSTCTRL	CLKTRCTRL	HW_AUTO
PM_IVA_PWRSTCTRL	POWERSTATE	OFF

33.3.4.4 Clocking Configuration

The ROM code detects the system input clock frequency (SYS_CLK1) from the sysboot[9:8] pins value. The supported system frequencies in the device are:

- 19.2 MHz
- 20 MHz
- 27 MHz

See Section 33.2.3.2, *Clocking scheme*, and Section 33.2.4.2, *System clock speed configuration*.

After detecting the input clock, the ROM code configures the clocks and DPLLs required for ROM code execution.

The configured DPLLs are:

- DPLL_PER: locked to provide clocks to peripheral blocks
- DPLL_CORE: locked to provide L3_MAIN interconnect, L4 interconnect, and EMIF clocks
- DPLL_MPU: locked
- DPLL_USB_OTG_SS/DPLL_USB DPLLs: Locked only in case of USB peripheral booting. It is left untouched otherwise.
- DPLL_SATA: locked only in the case of the SATA memory booting. It is left untouched otherwise.

The DPLLs and PRCM clock dividers are configured with the default values of the ROM code (depending on the detected system input clock) after cold or warm reset in order to give the same working conditions to the ROM code sequence.

Table 33-17 summarizes the default ROM code clock settings.

Table 33-17. ROM Code Default Clock Settings

Clock	Frequency (MHz)	Source
DPLL_CORE clock with F_{DPLL} locked frequency	2128	Gated SYS_CLK1
EMIF_PHY_GCLK[(1)]	44.33	DPLL_DDR (M2)
EMIF_DLL_GCLK	266	DPLL_DDR .HSDIVIDER (H11)
CORE_X2_CLK	266	DPLL_CORE.HSDIVIDER (H12)
CORE_CLK	266	CORE_X2_CLK
CORE_USB_OTG_SS_LFPS_TX_CLK	34.3	DPLL_CORE.HSDIVIDER (H13)
CORE_GPU_CLK	212.8	DPLL_CORE.HSDIVIDER (H14)
CORE_IPU_ISS_BOOST_CLK	212.8	DPLL_CORE.HSDIVIDER (H22)
CORE_ISS_MAIN_CLK	152	DPLL_CORE.HSDIVIDER (H23)
BB2D_GFCLK	177.3	DPLL_CORE.HSDIVIDER(H24)
L3_ICLK	133	CORE_CLK
L4_ICLK	66.5	L3_ICLK
MPU_DPLL_HS_CLK	266	CORE_X2_CLK
IVA_DPLL_HS_CLK	266	CORE_X2_CLK
DPLL_PER – clock with Fdpll locked frequency	768	Gated SYS_CLK1
FUNC_192M_CLK	192	DPLL_PER (M2)
FUNC_256M_CLK	256	DPLL_PER.HSDIVIDER (H11)
DSS_GFCLK	192	DPLL_PER.HSDIVIDER (H12)
PER_QSPI_CLK	192	DPLL_PER.HSDIVIDER(H13)
PER_GPU_CLK	192	DPLL_PER.HSDIVIDER (H14)
DPLL_MPU - clock with Fdpll locked frequency	2200	Gated SYS_CLK1
MPU_DPLL_CLK	588	PRM

(1) This clock is intentionally set up at low frequency to ensure correct initialization of external DRAM components.

Table 33-17. ROM Code Default Clock Settings (continued)

Clock	Frequency (MHz)	Source
MPU_GCLK	588	DPLL_MPU (M2)
DPLL_USB - clock with Fdpll locked frequency [(2)]	960	Gated SYS_CLK1
L3INIT_480M_GFCLK	480	DPLL_USB (M2)
L3INIT_60M_GFCLK	60	L3INIT_480M_GFCLK
DPLL_USB_OTG_SS - clock with F_{DPLL} locked frequency [(2)]	2500	Gated SYS_CLK1
DPLL_SATA - clock with Fdpll locked frequency [(2)]	1500	Gated SYS_CLK1

(2) This clock is locked specifically if USB peripheral or SATA memory booting is selected.

However it is possible to override the default clock settings. There are three ways to change DPLLs and all related clock divider, gating, and multiplexer configurations during the boot:

- ROM code default settings, described in Table 33-17. They are always applied at any reset.
- Software booting configuration after a software reset, described in Section 33.3.4.6, *Software Booting Configuration*
- The CH, described in Section 33.3.8.2, *Configuration Header*. The CH lets users have a known configuration (about GPMC and clock registers) after memory or peripheral booting. This configuration can be blocked by the software booting configuration. For more information, see Table 33-19, *Software Booting Configuration Strcuture*.

33.3.4.5 Booting Device List Setup

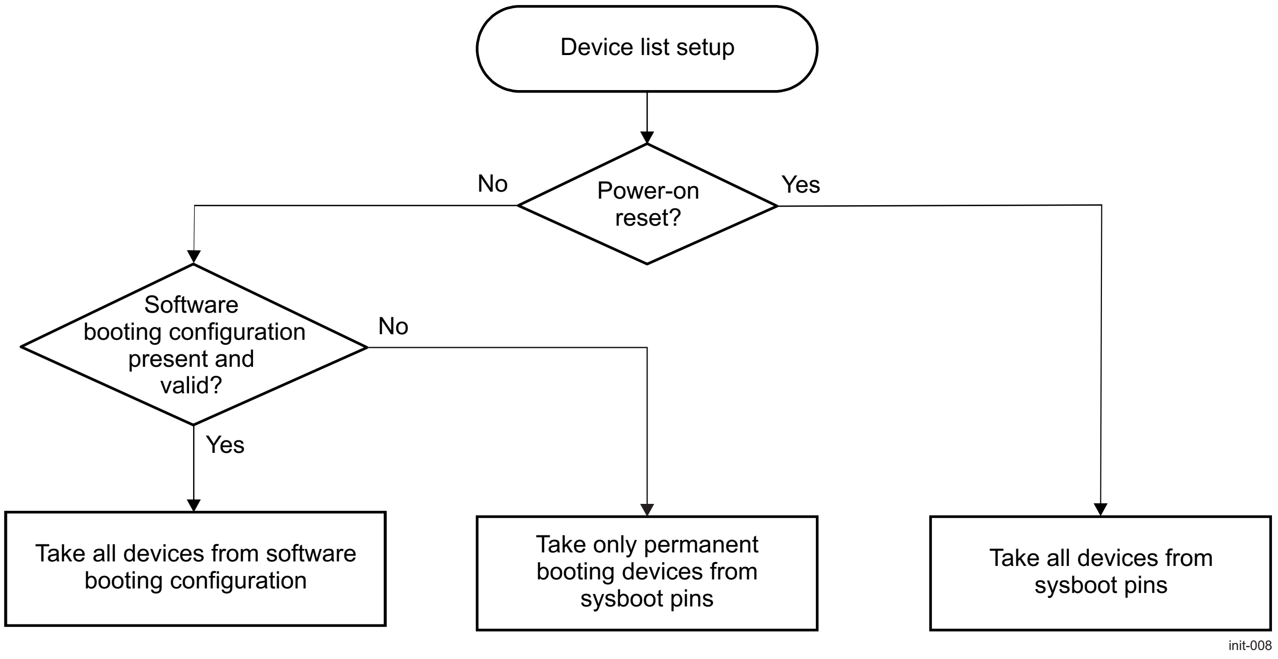
The ROM code creates a device list based on two sources:

- The software booting configuration (if present after a warm reset). The software booting configuration structure is stored in nonvolatile RAM SAR memory.
- The sysboot[5:0] signals latched in the control module are used to index the device table from which the list of devices is extracted.

Figure 33-9 shows how the ROM code sets up the device list depending on the reset source.

NOTE: Only permanent booting devices are put to the list when the reset is not power-on and the devices are taken from the sysboot pins. Users can force peripheral booting after software reset by using the software booting configuration.

Figure 33-9. Device List Setup


Device list setup
Power-on reset?
No
Yes
Software booting configuration present and valid?
No
Yes
Take all devices from software booting configuration
Take only permanent booting devices from sysboot pins
Take all devices from sysboot pins
init-008

33.3.4.6 Software Booting Configuration

The software booting configuration is a logical structure stored in SAR RAM memory, which is not cleared after warm resets or wakeups from low-power modes.

33.3.4.6.1 Public Use of SAR RAM

The public ROM code offers some flexibility about the location of the software booting configuration structure. The PUBLIC_SW_BOOT_CFG_ADDR pointer defines the start address of the structure within the SAR RAM bank (see Table 33-18).

Table 33-18. Public Use of SAR RAM

Logical name	Address	Size (Bytes)	Description
PUBLIC_SW_BOOT_CFG_ADDR	0x4AE2 6B00	4	Public software booting configuration pointer. The word value at this location is: - Reset (zero) on a cold reset. - Read on a warm reset.
PUBLIC_SAR_RAM_1_FREE	0x4AE2 6CC8	...	Recommended address for start of software booting configuration structure described in Section 33.3.4.6.2, *Software Booting Configuration Structure*.
PUBLIC_SAR_RAM_1_END	0x4AE2 6FFF	...	End of SAR RAM bank 1

As mentioned previously, the software booting configuration feature is optional. Hence, the public ROM code decides to use the feature based on the value read on a warm reset at the address pointed to by the PUBLIC_SW_BOOT_CFG_ADDR pointer. If the value matches the range 0x4AE2 6CC8 to 0x4AE2 6FFC, the ROM code extracts the structure at that address. The value pointed to by PUBLIC_SW_BOOT_CFG_ADDR is always overwritten to 0 on a cold reset.

It is possible to use the public SAR RAM area for any other purpose, such as storing traces for HLOS use. Obviously, care must be taken not to overwrite the locations used for low-power modes and/or software booting configuration, if used.

33.3.4.6.2 Configuration Structure for Software Booting

Table 33-19 describes the configuration structure for software booting . It offers three levels of flexibility for redefining the ROM code defaults after a warm reset:

- Redefining the default device booting list (from ROM code defaults given by SYSBOOT pins configuration)
- Redefining the default clock settings
- Redefining time-out linked to the peripheral booting mechanism

NOTE: The sections are provided as a linked list; therefore, the order and number of items is not relevant.

The ROM code searches for the next section at the location based on the size filled in the previous section. The clock configuration from software booting configuration may override the CH settings, if the appropriate bit is set in the configuration structure for software booting.

Table 33-19. Software Booting Configuration Structure

Field	Size (Bytes)	Description
Booting Configuration		
Section 1 key	4	Synchronization key for section 1: 0xCF00 AA01
Section 1 size	4	Size of section 1: 0x0000 000C (12)
Flags	2	Bits [4:1]: Mask the CH; when a bit is set to 1, a CH item is not analyzed: [1]: SETTINGS [3]: FLASH (GPMC) [4]: MMCSD [6]: QSPI
First device	2	Devices to be put into the device list: 0x01: XIP 0x02 XIP (with wait monitoring) 0x03: NAND 0x05: SD card 0x06: eMMC (boot partition, BP1 or BP2) 0x07: eMMC 0x09: SATA 0x0A: QSPI_1 0x0B: QSPI_4 0x43: UART 0x45: USB internal transceiver only 0x46: Other values: Reserved
Second device	2	
Third device	2	
Padding	2	Reserved
Clock Settings		
Section 2 key	4	Synchronization key for section 2: 0xCF00 AA02
Section 2 size	4	Size of section 2: 0x0000 008C (140)
Flags	4	Bit mask of various switches, active when set to 1: Bit [0]: Clock configuration defined in this structure is applied. Bit [1]: Reserved Bit [2]: Apply general clock settings. Bit [3]: Set and lock DPLL_PER. Bit [4]: Set and lock DPLL_MPU. Bit [5]: Set and lock DPLL_CORE. Bit [6]: Set and lock DPLL_USB (USB HS DPLL).

Table 33-19. Software Booting Configuration Structure (continued)

Field	Size (Bytes)	Description
		Bit [7]: Bypass DPLL_PER before setting clocks. Bit [8]: Bypass DPLL_MPU before setting clocks. Bit [9]: Bypass DPLL_CORE before setting clocks. Bit [10]: Bypass DPLL_USB (USB HS DPLL) before setting clocks. Others: Reserved
General Clock Settings		
CM_CLKSEL_CORE	4	Register value
CM_BYPCLK_DPLL_MPU	4	Register value
CM_BYPCLK_DPLL_IVA	4	Register value
CM_MPU_MPU_CLKCTRL	4	Register value
CM_CLKSEL_USB_60MHZ	4	Register value
MPU DPLL Settings		
CM_CLKMODE_DPLL_MPU	4	Register value
CM_AUTOIDLE_DPLL_MPU	4	Register value
CM_CLKSEL_DPLL_MPU	4	Register value
CM_DIV_M2_DPLL_MPU	4	Register value
Core DPLL Settings		
CM_CLKMODE_DPLL_CORE	4	Register value
CM_AUTOIDLE_DPLL_CORE	4	Register value
CM_CLKSEL_DPLL_CORE	4	Register value
CM_DIV_M2_DPLL_CORE	4	Register value
CM_DIV_M3_DPLL_CORE	4	Register value
CM_DIV_H11_DPLL_CORE	4	Register value
CM_DIV_H12_DPLL_CORE	4	Register value
CM_DIV_H13_DPLL_CORE	4	Register value
CM_DIV_H14_DPLL_CORE	4	Register value
CM_DIV_H21_DPLL_CORE	4	Register value
CM_DIV_H22_DPLL_CORE	4	Register value
CM_DIV_H23_DPLL_CORE	4	Register value
CM_DIV_H24_DPLL_CORE	4	Register value
PER DPLL Settings		
CM_CLKMODE_DPLL_PER	4	Register value
CM_AUTOIDLE_DPLL_PER	4	Register value
CM_CLKSEL_DPLL_PER	4	Register value
CM_DIV_M2_DPLL_PER	4	Register value
CM_DIV_M3_DPLL_PER	4	Register value
CM_DIV_H11_DPLL_PER	4	Register value
CM_DIV_H12_DPLL_PER	4	Register value
CM_DIV_H13_DPLL_PER	4	Register value
CM_DIV_H14_DPLL_PER	4	Register value
USB DPLL Settings		
CM_CLKMODE_DPLL_USB	4	Register value
CM_AUTOIDLE_DPLL_USB	4	Register value
CM_CLKSEL_DPLL_USB	4	Register value
CM_DIV_M2_DPLL_USB	4	Register value
Peripheral Booting Time-out Configuration		
Section 3 key	4	Synchronization key for section 3: 0xCF00 AA03

Table 33-19. Software Booting Configuration Structure (continued)

Field	Size (Bytes)	Description
Section 3 size	4	Size of the section 3: 0x0000 0008 (8)
Flags	2	Bit [0]: If cleared (= 0), the time-out USB field is ignored. Bit [1]: If cleared (= 0), the time-out boot message field is ignored. Other bits: Reserved
Time-out USB	2	Maximum time allowed for the host to complete the USB enumeration. • 0x0000: device waits indefinitely for the host to complete USB enumeration. • Others: The value expressed in milliseconds defines the maximum time the device will wait for the host to complete the USB enumeration (maximum 65.5 seconds).
Time-out boot message	2	Maximum time allowed for the host to send a booting message. • 0x0000: device waits indefinitely for the host to send a booting message. • Others: The value expressed in milliseconds defines the maximum time the device will wait for the host to send a booting message (maximum 65.5 seconds).
Padding	2	

33.3.5 Peripheral Booting

33.3.5.1 Description

The ROM code can boot from these peripherals:

- USB1: High-, and Full-speed USB from USB1 internal transceivers
- UART3: 115.2 Kbps, 8 bits, even parity, 1 stop-bit, no flow control

The purpose of booting from a peripheral is to download a flash loader code from an external host. This booting method is used primarily for programming flash memories connected to the device (for example, in the case of initial flashing, firmware update or servicing). Figure 33-12 shows the overall peripheral booting procedure. It consists of a synchronization phase (handshake between the host and the device) and a transfer phase. The synchronization phase is similar for UART and USB boots. Both transfer phases use the same procedure.

When booting from the UART, the ROM code first initializes the UART3 interface. Then the ROM code sends an ASIC ID block of data. From there, it expects to receive a boot message from the host within 300 ms, by default. Figure 33-10 shows this procedure.

Figure 33-10. Synchronization Phase for UART

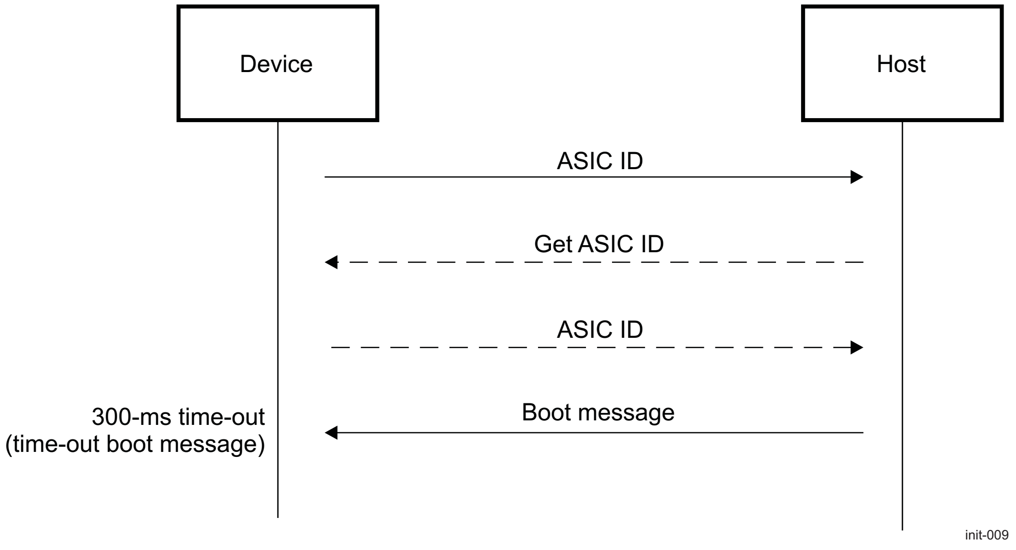
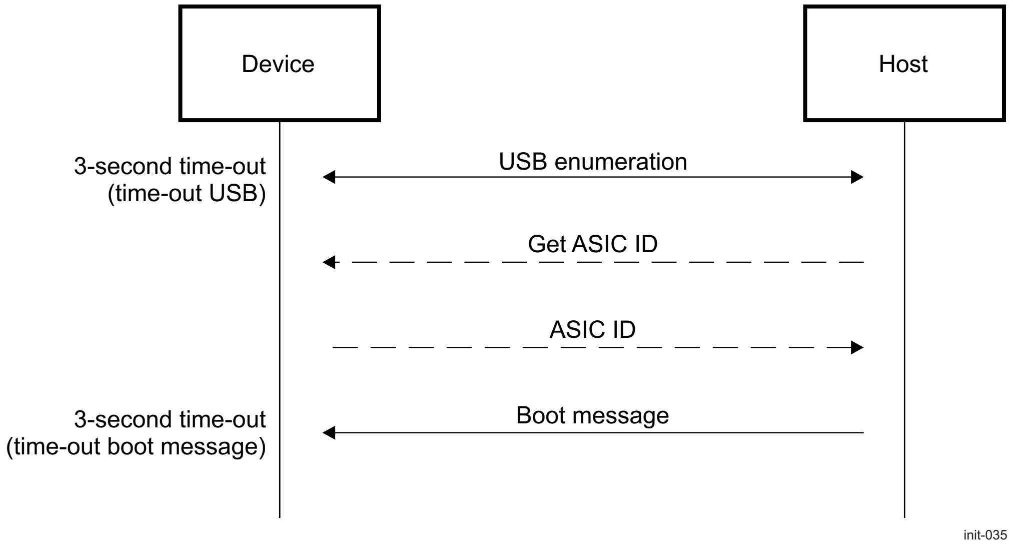


Figure 33-11 shows the procedure when booting from the USB.

Figure 33-11. Synchronization Phase for USB



During the synchronization phase (see Figure 33-10 and Figure 33-11), the device can provide a small packet of data called the ASIC ID (described in Table 33-20). It is a simple structure that contains different kinds of information, such as ROM version, checksums, and ID.

The default time-outs involved during the synchronization phase can be redefined after a warm reset by means of the software booting configuration (described in Section 33.3.4.6, *Software Booting Configuration*). The host can decide the desired operation by providing a booting message (see Table 33-24). This message can be: Get ASIC ID, peripheral boot, change device, or next device. If the device receives the Get ASIC ID boot message, it sends back the ASIC ID contents.

If the change device or next device message is received, the ROM code stops the current peripheral booting procedure and returns to the main booting, which decides about the next booting device according to the boot message received.

If the peripheral boot message is received without a time-out, the device is entering the transfer phase. From there, the flash loader image size (as a 32-bit word) and the flash loader image itself are expected to be received. The ROM code waits up to 1 minute for completion of image size reception, and up to 1 more minute to download the image. If the download procedure does not complete before this time, the peripheral booting procedure aborts. ROM code continues to examine the devices included in the booting device list. If the download procedure passes, then the image can be executed.

The flash loader image is downloaded directly into internal RAM from address 0x4030 0000 and the maximum size of the downloaded image is 504 KiB.

NOTE: Sending an image size of zero skips the peripheral booting procedure.

The USB or UART connection is left open at the end of the transfer phase and once exiting the ROM code for the initial software to take over. It means the initial software can reuse the currently established connection. In the case of a USB connection, the endpoints can be reused as such, without closing the connection and performing a full enumeration again.

Table 33-20. ASIC ID Structure

ASIC ID Item	Size (Bytes)	Description
Items	1	Number of subblocks
ID subblock	7	Device identification information
Reserved subblock	4	Reserved
Reserved subblock	23	Reserved
Reserved subblock	35	Reserved

Table 33-20. ASIC ID Structure (continued)

ASIC ID Item	Size (Bytes)	Description
Checksum subblock	11	CRC (4 bytes)

Table 33-21. Items

Offset	Size (Bytes)	Description
0x00	1	0x05: Number of subblocks USB 0x04: Number of subblocks UART[1]

[1] The checksum subblock is not transmitted over the UART.

Table 33-22. ID Subblock

Offset	Size (Bytes)	Description
0x01	1	0x01: Subblock ID
0x02	1	0x05: Subblock size
0x03	1	0x01: Fixed value
0x04	2	0x56, 0x41: Device ID number
0x06	1	0x07: CH enabled 0x17: CH disabled
0x07	1	ROM code revision 0x01: SR1.0 0x02: SR1.1 and SR2.0

Table 33-23. Checksum Subblock[1]

Offset	Size (Bytes)	Description
0x46	1	0x15: Subblock ID
0x47	1	0x09: Subblock size
0x48	1	0x01: Fixed value
0x49	4	ROM CRC
0x4D	4	0x0000 0000:

[1] The checksum subblock is not transmitted over the UART.

Table 33-24. Booting Messages

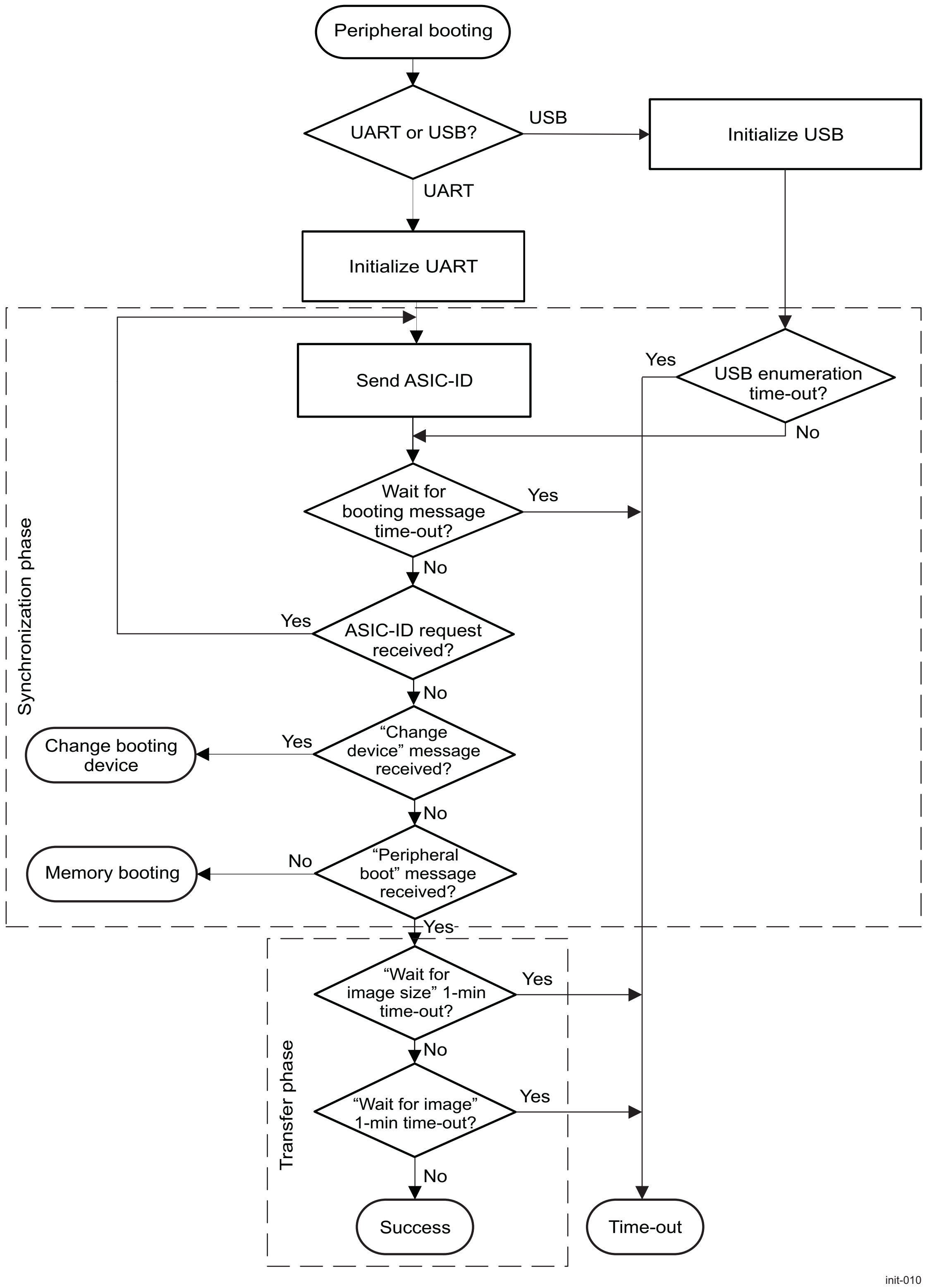
Message Name	Value	Description
Peripheral boot	0xF003 0002	Continue peripheral booting.
Get ASIC ID	0xF003 0003	ASIC ID request. The Get ASIC ID request message is optional. If received, the ROM code sends its ASIC ID data to the host in return. The host can issue the Get ASIC ID message multiple times if required. Table 33-20 describes the structure of the ASIC ID.
Change device	0xF003 xx06	Skip current peripheral booting and continue booting from device type indicated by xx: 0x01: XIP 0x02: XIP (with wait monitoring) 0x03: NAND 0x05: SD card 0x06: eMMC (from boot partition BP1 or BP2) 0x07: eMMC 0x09: SATA 0x0A: QSPI_1 0x0B: QSPI_4 0x43: UART

Table 33-24. Booting Messages (continued)

Message Name	Value	Description
		0x45: USB (from internal transceiver) Others: Reserved
Next device	0xFFFF FFFF	Skip current device and move to the next device on the device list.
Memory booting	Others	Skip current peripheral booting and move to the first device for memory booting.

Figure 33-12 shows the peripheral booting procedure.

Figure 33-12. Peripheral Booting Procedure


Peripheral booting
UART or USB?
USB
Initialize USB
UART
Initialize UART
Send ASIC-ID
Yes
USB enumeration time-out?
No
Wait for booting message time-out?
Yes
No
Yes
ASIC-ID request received?
No
Synchronization phase
Change booting device
Yes
"Change device" message received?
No
Memory booting
No
"Peripheral boot" message received?
Yes
"Wait for image size" 1-min time-out?
Yes
No
Transfer phase
"Wait for image" 1-min time-out?
Yes
No
Success
Time-out
init-010

33.3.5.2 Initialization Phase for UART Boot

The ROM code supports booting from a UART interface with the following characteristics:

- UART3 interface
- Communication parameters set to 115.2 Kbps, 8 bits, even parity, 1 stop-bit, no flow control
- Two-pin interface: RX/TX
- The boot message default time-out is 300 ms (time-out boot message)

33.3.5.3 Initialization Phase for USB Boot

The ROM code supports booting from a USB interface with the following characteristics:

- Using the USB1 subsystem in USB2.0 mode
- Device integrated USB transceiver (USB2PHY1)
- Enumeration default time-out is 3 seconds (time-out USB)
- The boot message default time-out is 3 seconds (time-out boot message).

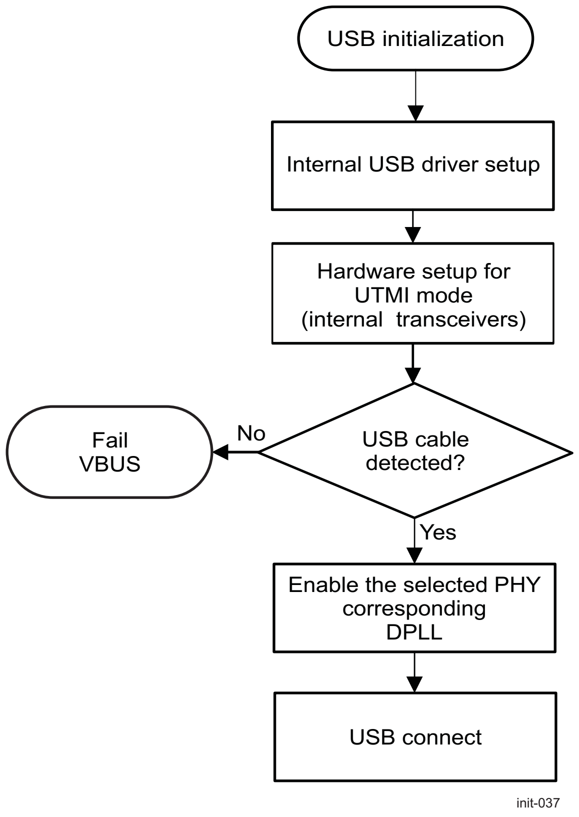
NOTE: The ROM code does not handle any OTG-specific features.

33.3.5.3.1 Initialization Procedure

NOTE: The internal USB2PHY1 (HS) transceiver must be powered and configured upon startup. No action is expected from the device ROM code for its configuration.

Figure 33-13 shows the USB initialization procedure.

Figure 33-13. USB Initialization Procedure


USB initialization
Internal USB driver setup
Hardware setup for UTMI mode (internal transceivers)
USB cable detected?
No
Fail VBUS
Yes
Enable the selected PHY corresponding DPLL
USB connect
init-037

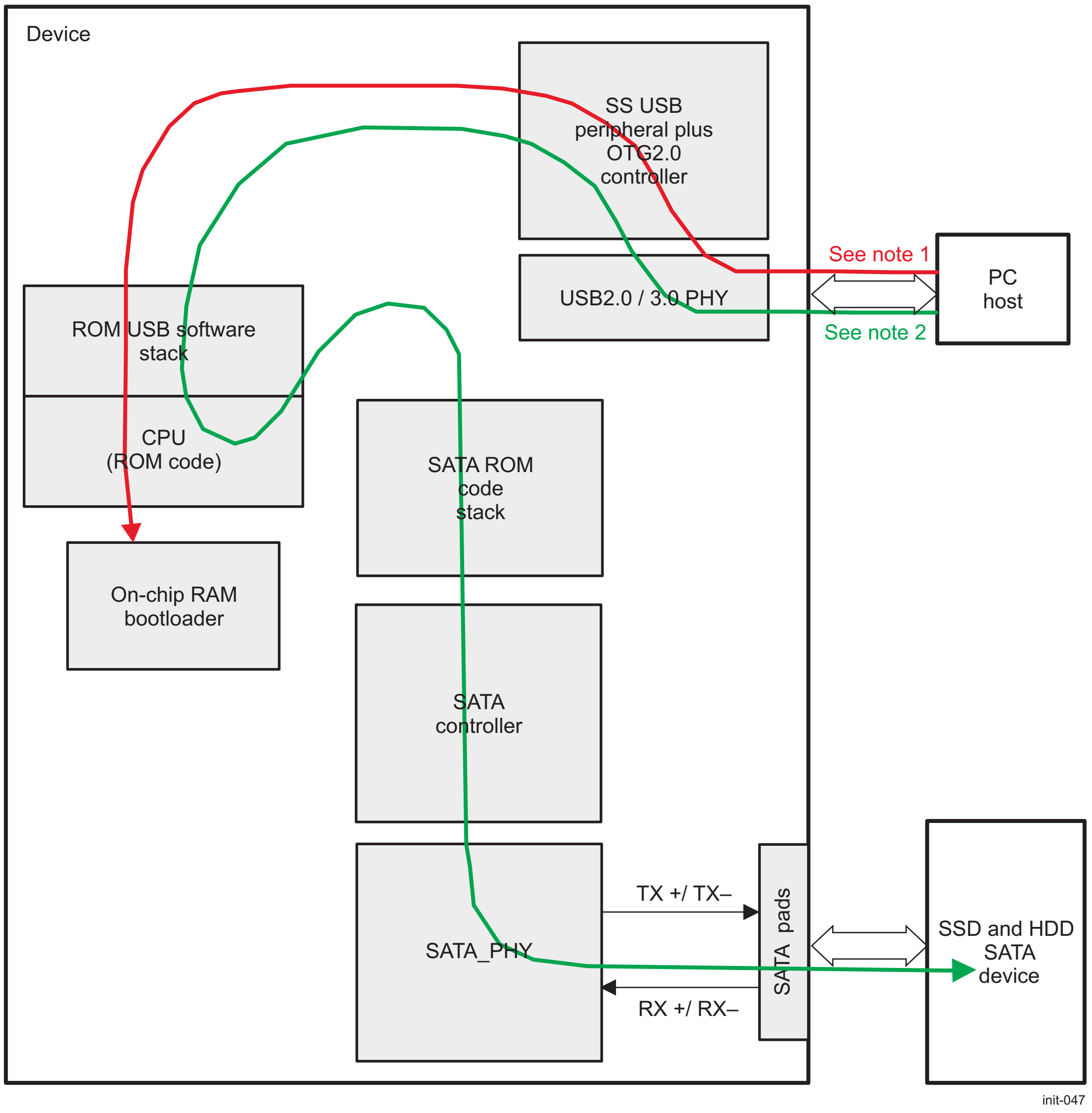
NOTE: The ROM code does not consider the USB PHYs charger detection circuitry status.

NOTE: The ROM code assumes that VBUS is always present.

33.3.5.3.2 SATA Peripheral Device Flashing over USB Interface

In the next scheme (see Figure 33-14), it is assumed that the new version of the firmware Flash Loader Host is located in the PC host. In the same way, the application loaded to device internal L3 On-chip RAM (using the peripheral boot function of the device ROM code) is called Flash Loader Second.

Figure 33-14. SATA Flashing Over USB



(1) ROM boots from USB controller and downloads deviceFlash Second in RAM.

(2) The PC host is allowed to flash the device firmware located in the SATA device.

33.3.5.3.3 USB Driver Descriptors

USB devices use descriptors to report their attributes. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type. Using descriptors allows concise storage of the attributes of individual configurations so that each configuration can reuse descriptors or portions of descriptors from other configurations that have the same characteristics. Where appropriate, descriptors contain references to string descriptors. String descriptors contain displayable, human-readable description information. These descriptor details can be used for tool development or debugging:

- Device descriptor

  A device descriptor contains general information about a USB device, including global information that applies to the device and all device configurations. A USB device has only one device descriptor. A device-qualifier descriptor is required because the ROM code uses the HS feature of the USB core. Table 33-25 lists the device descriptors.

Table 33-25. Device Descriptor

Field	Value	Description
bLength	0x12	Size of this descriptor in bytes
bDescriptorType	0x01	Device descriptor type
bcdUSB	0x0210	USB specification release number in binary coded decimal (BCD) format
bDeviceClass	Vendor-specific (0xFF)	Class code
bDeviceSubClass	Vendor-specific (0xFF)	Subclass code
bDeviceProtocol	Vendor-specific (0xFF)	Protocol code
bMaxPacketSize0	0x40	Maximum packet size for endpoint 0
idVendor	0x0451	Vendor ID (Texas Instruments), TI default value
idProduct	0xD013	Product ID, TI default value
bcdDevice	0x0000	Device release number
iManufacturer	See Section 33.3.5.3.4.	Index of string descriptor describing manufacturer
iProduct	See Section 33.3.5.3.4.	Index of string descriptor describing product
iSerialNumber	See Section 33.3.5.3.4.	Index of string descriptor describing device serial number
bNumConfigurations	0x01	Number of possible configurations

- Device-qualifier descriptor

  The device-qualifier descriptor contains information about a HS-capable device that changes if the device operates at its other speed. This descriptor is retrieved by the host using the GetDescriptor() request (standard device request). Table 33-26 lists the device-qualifier descriptors.

Table 33-26. Device-Qualifier Descriptor

Field	Value	Description
bLength	0x0A	Size of this descriptor in bytes
bDescriptorType	0x06	Device-qualifier descriptor type
bcdUSB	0x0210	USB specification release number in BCD
bDeviceClass	0xFF	Class code
bDeviceSubClass	0xFF	Subclass code
bDeviceProtocol	0xFF	Protocol code
bMaxPacketSize0	0x40	Maximum packet size for endpoint 0
bNumConfigurations	0x01	Number of possible configurations
bReserved	0x00	Reserved for future use

- Configuration descriptor

  This descriptor gives information about a specific device configuration. It describes the number of interfaces supported by the configuration (see Table 33-27).

Table 33-27. Configuration Descriptor

Field	Value	Description
bLength	0x09	Size of this descriptor in bytes
bDescriptorType	0x02	Configuration descriptor type
wTotalLength	–	Combined length of all descriptors
bNumInterfaces	0x01	Number of interfaces supported
bConfigurationValue	0x01	Value to use as an argument for the SetConfiguration() request
iConfiguration	Index	Index of string descriptor describing this configuration
bmAttributes	0xC0	Power setting and remote wakeup
bMaxPower	0x32	Maximum power consumption of the USB device

- Other speed configuration descriptor

  This descriptor describes the configuration of a HS-capable device if it operates at its other possible speed (see Table 33-28).

Table 33-28. Other Speed Configuration Descriptor

Field	Value	Description
bLength	0x09	Size of this descriptor in bytes
bDescriptorType	0x07	Other speed configuration descriptor type
wTotalLength	–	Combined length of all descriptors
bNumInterfaces	0x01	Number of interfaces supported
bConfigurationValue	0x01	Value to use as an argument for the SetConfiguration() request
iConfiguration	Index	Index of string descriptor describing this configuration
bmAttributes	0xC0	Power setting and remote wakeup
bMaxPower	0x32	Maximum power consumption of the USB device

- Interface descriptor

  This descriptor describes a specific interface in a configuration (see Table 33-29).

Table 33-29. Interface Descriptor

Field	Value	Description
bLength	0x09	Size of this descriptor in bytes
bDescriptorType	0x04	Interface descriptor type
bInterfaceNumber	0x00	Number of this descriptor
bAlternateSetting	0x00	Value to select the alternate setting
bNumEndpoints	0x02	Number of endpoints used for this interface
bInterfaceClass	0xFF	Class code
bInterfaceSubClass	0xFF	Subclass code
bInterfaceProtocol	0xFF	Protocol code
iInterface	Index	Index of string descriptor describing this interface

- Endpoint descriptor

  Each endpoint used for an interface has its own descriptor. This descriptor contains information required by the host to determine the bandwidth requirements of each endpoint. This descriptor is returned as part of the GetDescriptor(Configuration) request (see Table 33-30 and Table 33-31).

Table 33-30. BULK IN Endpoint Descriptor

Field	Value	Description
bLength	0x07	Size of this descriptor in bytes
bDescriptorType	0x05	Endpoint descriptor type

Table 33-30. BULK IN Endpoint Descriptor (continued)

Field	Value	Description
bEndpointAddress	0x81 (1 IN)	Address of the endpoint on the USB device
bmAttributes	0x02 (Bulk)	Type of transfer
wMaxPacketSize	See [(1)].	Number of endpoints used for this interface
bInterval	0x00	Maximum NAK rate

(1) The maximum size is 0x0200 (512 bytes) for HS bulk endpoint and 0x0040 (64 bytes) for FS bulk endpoint.

Table 33-31. BULK OUT Endpoint Descriptor

Field	Value	Description
bLength	0x07	Size of this descriptor in bytes
bDescriptorType	0x05	Endpoint descriptor type
bEndpointAddress	0x01 (1 OUT)	Address of the endpoint on the USB device
bmAttributes	0x02 (Bulk)	Type of transfer
wMaxPacketSize	See [(1)].	Number of endpoints used for this interface
bInterval	0x00	Maximum NAK rate

(1) The maximum size is 0x0200 (512 bytes) for HS bulk endpoint and 0x0040 (64 bytes) for FS bulk endpoint.

- String descriptors

  String descriptors use UNICODE encoding. The strings in a USB device can support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a 16-bit language ID (LANGID) defined by the USB interface. String index 0 for all languages returns a string descriptor that contains an array of 2-byte LANGID codes supported by the device.

  For the description of the different string descriptors, see:

  – The language ID string descriptor (Table 33-32)
  – The manufacturer ID string descriptor (Table 33-33)
  – The product ID string descriptor (Table 33-34)
  – The configuration string descriptor (Table 33-35)
  – The interface string descriptor (Table 33-36)

Table 33-32. Language ID String Descriptor

Field	Value	Description
bLength	0x04	Size of this descriptor in bytes
bDescriptorType	0x03	String descriptor type
wLangId	0x0409 (US English)	Language ID code

Table 33-33. Manufacturer ID String Descriptor

Field	Value	Description
bLength	0x24	Size of this descriptor in bytes
bDescriptorType	0x03	String descriptor type
bString	Texas Instruments	Manufacturer string

Table 33-34. Product ID String Descriptor

Field	Value	Description
bLength	0x12	Size of this descriptor in bytes
bDescriptorType	0x03	String descriptor type
bString	<string>	Product string

Table 33-35. Configuration String Descriptor

Field	Value	Description
bLength	0x08	Size of this descriptor in bytes
bDescriptorType	0x03	String descriptor type
bString	pbc	Configuration string

Table 33-36. Interface String Descriptor

Field	Value	Description
bLength	0x08	Size of this descriptor in bytes
bDescriptorType	0x03	String descriptor type
bString	pbi	Interface string

33.3.5.3.4 USB Customized Vendor and Product IDs

When device ROM Code enumerates through USB, one may want to use his/her own vendor and product IDs rather than TI defaults. Product and Vendor IDs are transmitted as part of the USB Standard Device Descriptor during the USB device enumeration process (resp. idProduct and idVendor 16 bits fields).

Product IDs (PIDs) and vendor IDs (VIDs) are stored as part of device chip eFuses and readable from control module. The built-in USB ROM driver behaves differently whether the VID value is either left unfused, or fused with a customer-specific VID. Thus, two main cases are distinguished:

- VID is fused with a customer-specific ID value
- VID is left unfused

Table 33-37 lists standard USB ROM device descriptors. Table 33-38 lists USB ROM descriptor strings.

Table 33-37. USB ROM Standard Device Descriptor

Device Descriptor Field	Size (Bytes)	VID≠0x0000[(1)] (fused)	VID=0x0000 (unfused)[(2)]
bLength	1	0x12	
bDescriptorType	1	0x1	
bcdUSB	2	0x0200 (if enumerated in USB2.0 HS or FS)	
bDeviceClass	1	0xFF	
bDeviceSubClass	1	0xFF	
bDeviceProtocol	1	0xFF	
bMaxPacketSize0	1	0x40 (64 bytes if enumerated in USB2.0 HS or FS)	
idVendor	2	(VID value from control module)	0x0451
idProduct	2	(PID value from control module)	0xD013
bcdDevice	2	0x0000	
iManufacturer	1	0x20	0x21
iProduct	1	0x24	0x25
iSerialNumber	1	0	
bNumConfigurations	1	1	

(1) VID other than 0x0 and VID other than 0xFFFF and VID other than 0x0451 values
(2) VID=0x0 or VID=0xFFFF or VID=0x0451

Table 33-38. USB ROM Descriptor Strings

String Descriptor	Size (bytes)	VID ≠ 0x0000	VID=0x0000(unfused)[(1)]
Serial Number		N/A	

(1) VID=0x0451

Table 33-38. USB ROM Descriptor Strings (continued)

String Descriptor	Size (bytes)	VID ≠ 0x0000	VID=0x0000(unfused)(1)
Configuration	8	"pbc"	
Interface	8	"pbi"	
Product	8/18	N/A	<string>
Manufacturer	8/36	N/A	"Texas Instruments"

The ROM code transmits additional descriptors as part of the enumeration procedure: configuration descriptor, device qualifier, language ID string, configuration string, interface string, and function string descriptors. Those do not depend on the VID or PID value.

33.3.5.3.5 USB Driver Functionality

- Transactions supported:
  - Control transactions: Used for standard device requests
  - Bulk transactions: Used for data transfer in the image downloading stage. The ASIC ID is sent to the BULK IN endpoint and the image is transferred from the host over the BULK OUT endpoint.

  The device USB device first attaches to the host as an FS device. In the reset mechanism, the USB core requests HS operation. If the HS negotiation in the reset phase succeeds, further transactions are at HS; otherwise, they are at FS. After reset, the USB driver checks for the speed of the device, whether it is FS or HS. Depending on the speed configured by the host, the standard USB device requests are answered with the corresponding descriptors.
- Standard device request restrictions:

  Some standard device requests are not supported by the driver because the USB driver is dedicated for use by the ROM code for peripheral booting. Table 33-39 lists the standard device requests supported by the driver.

Table 33-39. Standard Device Requests Supported

Request	Description	Support
CLEAR_FEATURE	Sets or clears a specific feature	Supported only for the ENDPOINT_HALT feature
GET_CONFIGURATION	Returns the current device configuration value	Yes
GET_DESCRIPTOR	Returns the specified descriptor	Yes
GET_INTERFACE	Returns the selected alternate setting for the specified interface	Yes
GET_STATUS	Returns the status for the specified recipient	Yes
SET_ADDRESS	Sets the device address	Yes
SET_CONFIGURATION	Sets the device configuration	Yes
SET_DESCRIPTOR	Updates existing descriptors or adds new descriptors	Runtime updating of descriptors is not supported.
SET_FEATURE	Sets or enables a specific feature	Supported only for the ENDPOINT_HALT feature
SET_INTERFACE	Selects an alternate setting in an interface	Runtime setting of alternate features is not supported.
SYNCH_FRAME	Sets and reports an endpoint synchronization frame	No, because isochronous transfers are not used

33.3.6 Fast External Booting

33.3.6.1 Overview

The fast external boot is a special memory booting mode. It consists of a blind jump to a code in an external XIP memory device connected to GPMC CS0 and lets customers create their own booting code. Fast external booting is selected by means of the SYSBOOT[5:0] pins .

The NOR device must fulfill the following requirements for fast XIP mode:

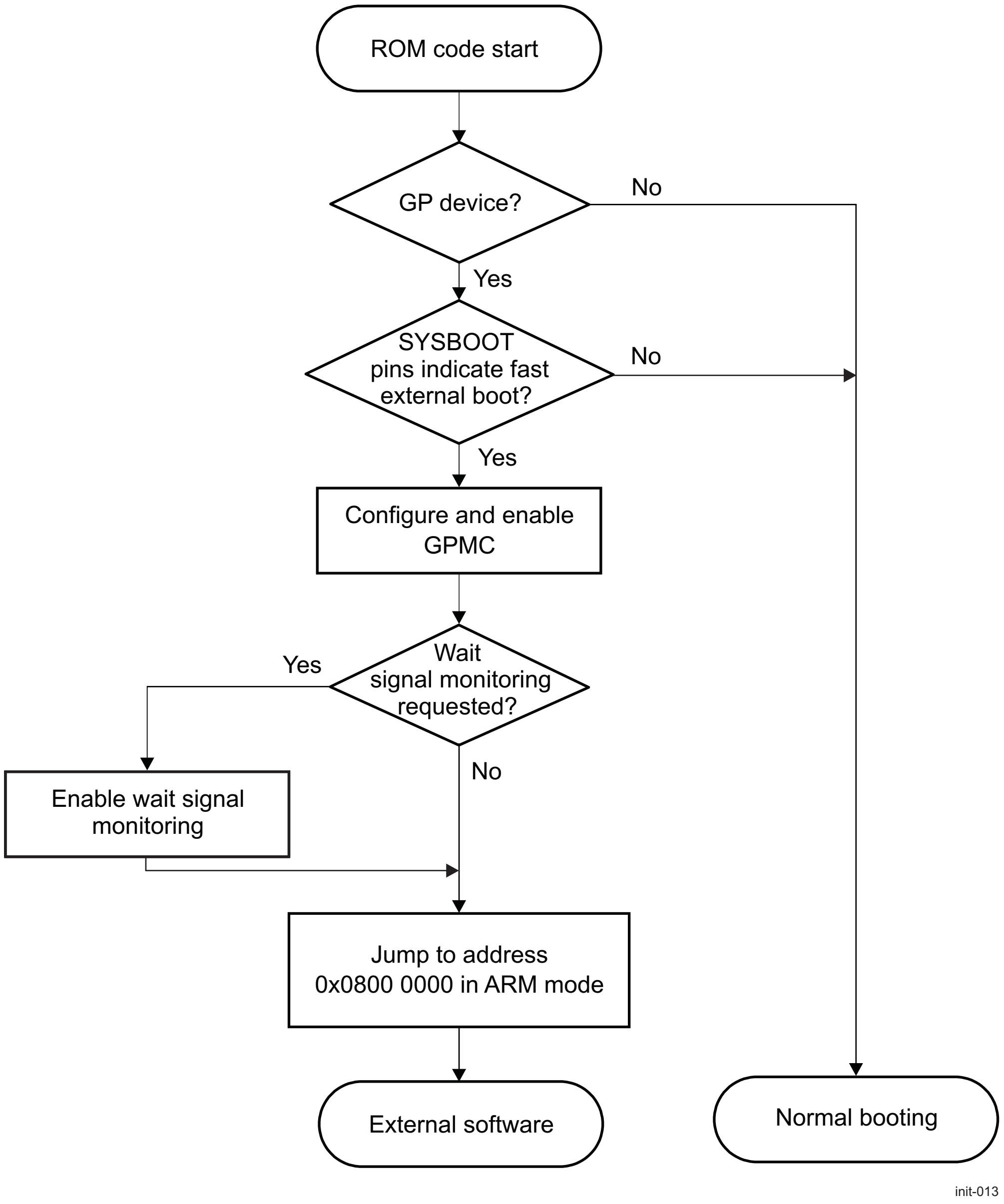
- Non-muxed or address/data multiplexed mode, configured through sysboot[12:11] (=0x1 for A/D mux)
- 8-bit or 16-bit data bus width, configured through sysboot[13] (=1 for 16-bit)
- CS0 chip select
- Device wait signal connected to the WAIT0 GPMC signal (if used)
- The wait monitoring enable/disable is based on the value of SYSBOOT[10]. SYSBOOT[10] value is exported to GPMC by hardware means.

The jump is performed with minimum on-chip ROM code execution.

33.3.6.2 Fast External Booting Procedure

Figure 33-15 shows the procedure for fast external boot. The code does not use any RAM and is designed for fast execution.

Figure 33-15. Fast External Boot Procedure


ROM code start
GP device?
No
Yes
SYSBOOT pins indicate fast external boot?
No
Yes
Configure and enable GPMC
Wait signal monitoring requested?
Yes
No
Enable wait signal monitoring
Jump to address 0x0800 0000 in ARM mode
External software
Normal booting
init-013

33.3.7 Memory Booting

33.3.7.1 Overview

The memory booting process starts an external code in memory devices. ROM code can use only the memory type of booting devices as permanent booting devices (that is, devices examined after both cold [POR] and warm resets). Temporary booting devices are examined only after cold resets. The supported permanent booting devices are:

- NOR flash devices
- NAND flash devices
- SPI/QSPI flash memories
- eMMC memories
- SD cards
- SATA SSD and HDD storage memory devices

Two main groups of permanent booting devices are distinguished by code shadowing. Code shadowing means copying code from a nondirectly addressable device (non-XIP) to RAM, where the code can be executed. Directly addressable devices are XIP devices.

Figure 33-16 shows the general memory booting procedure common to all types of devices. First, CH is copied to internal RAM. It is copied even for XIP devices, because the device can temporarily lose a connection with XIP memory during CH execution (for example, while updating interface timings). The second step is to shadow the image, if the device is not XIP. The last step is image execution and any return from image results in a dead loop.

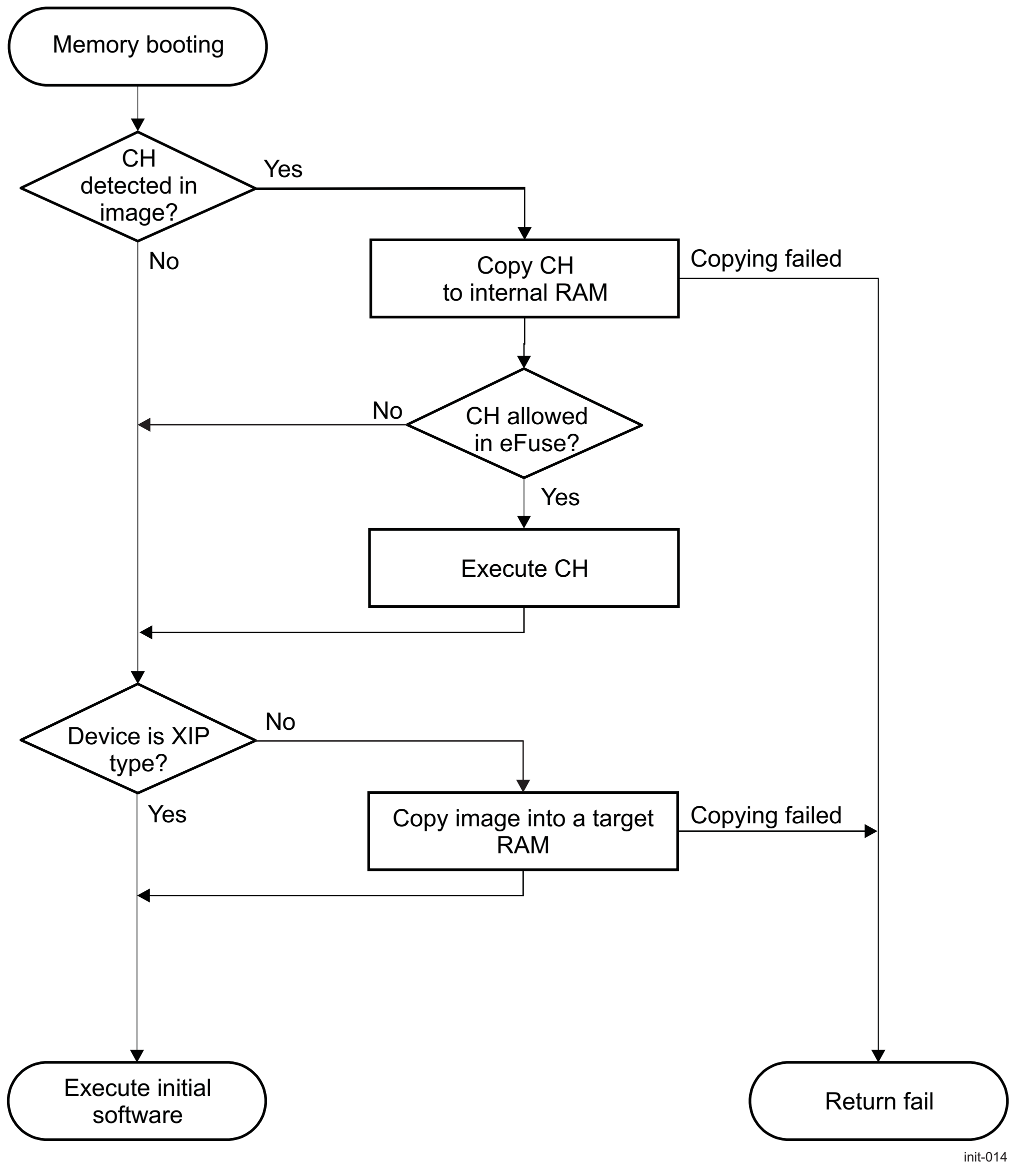
If CH copying or shadowing fails, memory booting returns to the main booting procedure, which selects the next device for booting.

NOTE: A booting image is considered to be present, when the first 4-byte word of the sector is not equal to 0000 0000h or FFFF FFFFh.

During the first read sector (512 bytes) call, sectors are copied to a temporary device on-chip SRAM buffer. Once the image is found and the destination address is known, the content of the temporary buffer is moved to the target device on-chip SRAM location so it is required to reread the first image sector. GP header is not copied into target buffer location; therefore, only executable code is in device on-chip RAM, with the first executable instruction at the destination address.

SATA, eMMC, SD cards, SPI/QSPI and NAND devices can hold up to four copies of the booting image. Therefore, the ROM code searches for one valid image out of the four copies, if present, by walking over the first blocks of mass storage space. Other XIP devices (NOR) use only one copy of the booting image.

Figure 33-16. Memory Booting Procedure

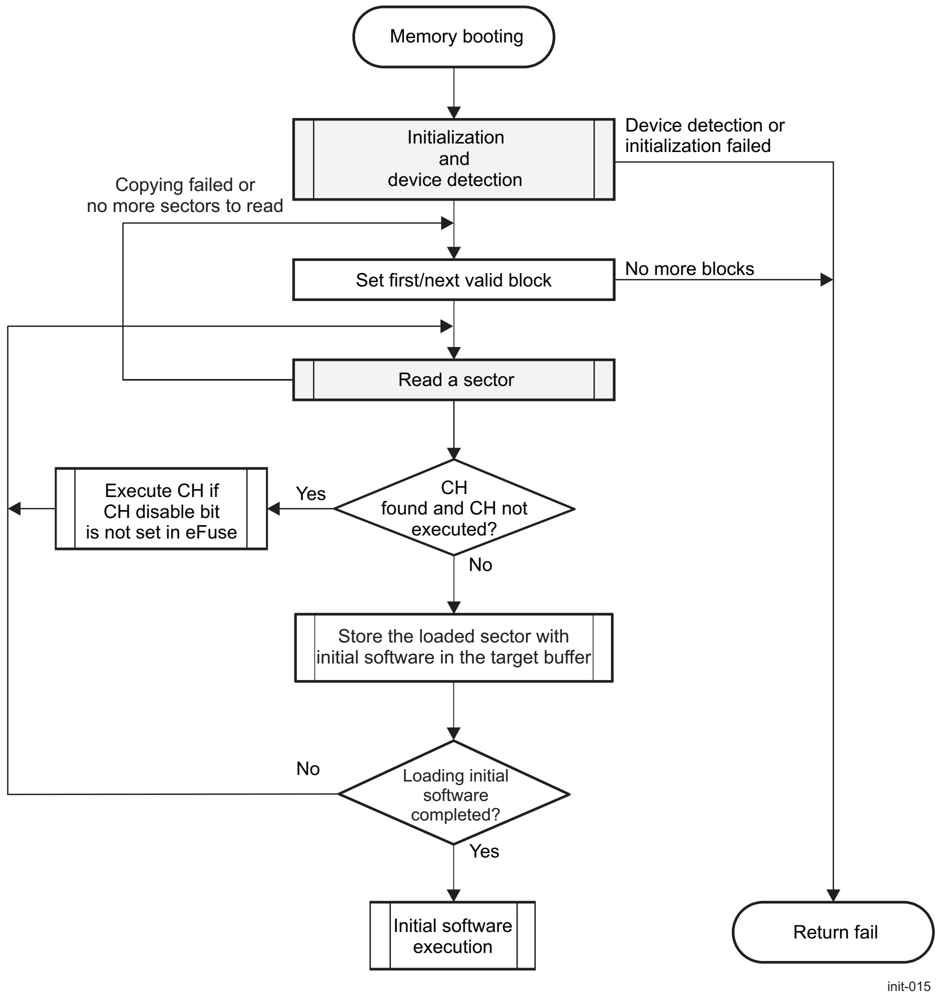


33.3.7.2 Non-XIP Memory

Figure 33-17 shows the procedure used when memory booting runs with non-XIP memories. The shaded procedures are specific to each memory. The ROM code searches for the image in the first four physical blocks of the memories. Other devices use only one copy of the image and the block loop runs only once.

During image shadowing, the CH is expected to be in a separate sector before the initial software.

Figure 33-17. Image Shadowing


Memory booting
Initialization and device detection
Device detection or initialization failed
Copying failed or no more sectors to read
Set first/next valid block
No more blocks
Read a sector
Execute CH if CH disable bit is not set in eFuse
Yes
CH found and CH not executed?
No
Store the loaded sector with initial software in the target buffer
No
Loading initial software completed?
Yes
Initial software execution
Return fail
init-015

For more information about the GPMC module, see Section 15.4, *General-Purpose Memory Controller*.

The following sections describe the supported device types.

33.3.7.3 XIP Memory

The ROM code can boot directly from XIP devices, such as NOR flash memories, that have the following characteristics:

- The GPMC is the communication interface.
- Memories up to 1Gibit (128MiB) can be connected.
- 8-bit or 16-bit data bus width, configured through sysboot[13] (=1 for 16-bit)
- Non-muxed or address/data multiplexed mode, configured through sysboot[12:11] (=0x1 for A/D mux)
- The GPMC clock is 133 MHz.
- The device is connected to CS0 mapped to address 0x0800 0000.
- The wait pin gpmc_wait0 signal is monitored according to the sysboot[10] configuration pin (=1 is enabled)
- For an XIP memory booting, no user intervention is required; the following debugging steps are described. Only the CH, which is not mandatory, lets users change clock settings and GPMC parameters. Failure in CH copying causes a return to the main booting procedure, which selects the next device for booting.

Booting from an XIP device consists of the following steps:

1. Configure the GPMC for XIP device access.
2. Verify that the CH is present at address 0x0800 0000. If the CH is present, copy the entire sector (512 bytes) to internal RAM and execute the CH.
3. Set the image location:
   - 0x0800 0000 if the CH is not found
   - 0x0800 0200 if the CH is found
4. Verify that a bootable image is at the image location.
5. If the image is found, execute it.
6. If the image is not found, return from XIP booting to the main booting loop.

33.3.7.3.1 GPMC Initialization

Table 33-40 lists the timing settings of the GPMC when set for XIP and other address-data accessible devices. Table 33-40 is included for debug information.

Table 33-40. XIP Timing Parameters

Parameter	Value (Clock Cycles)[1]	Register Initialization (where i = 0)	Reset Value
Write cycle period	17	The GPMC_CONFIG5_i[12:8] WRCYCLETIME bit field is set to 0x11.	0x11
Read cycle period	17	The GPMC_CONFIG5_i[4:0] RDCYCLETIME bit field is set to 0x11.	0x11
CS low time	1	The GPMC_CONFIG2_i[3:0] CSONTIME bit field is set to 0x1.	0x1
CS high time	16	The GPMC_CONFIG2_i[12:8] CSRDOFFTIME bit field is set to 0x10.	0x10
ADV low time	1	The GPMC_CONFIG3_i[3:0] ADVONTIME bit field is set to 0x1.	0x1
ADV high time	2	The GPMC_CONFIG3_i[12:8] ADVRDOFFTIME bit field is set to 0x2.	0x2
OE low time	3	The GPMC_CONFIG4_i[3:0] OEONTIME bit field is set to 0x3.	0x6
OE high time	16	The GPMC_CONFIG4_i[12:8] OEOFFTIME bit field is set to 0x10.	0x10
WE low time	3	The GPMC_CONFIG4_i[19:16] WEONTIME bit field is set to 0x3.	0x05
WE high time	15	The GPMC_CONFIG4_i[28:24] WEOFFTIME bit field is set to 0xF.	0x10
Data latch time	15	The GPMC_CONFIG5_i[20:16] RDACCESSTIME bit field is set to 0xF.	0x0F

[1] The one clock cycle is approximately 7.5 ns, which corresponds to a 133-MHz frequency.

There is no specific identification routine executed before booting from an XIP device.

33.3.7.4 NAND

NAND flash memory is not an XIP device; it requires shadowing before the code can be executed. ROM code support for the NAND flash devices has the following characteristics:

- The GPMC is the communication interface.
- Device from 512 Mibit (64 MiB) to 64 Gibit (8 GiB)
- ×8 and ×16 bus width
- Support for large page size (2048 bytes + 64 spare bytes) or very large page size (4096 bytes + 128/218 spare bytes)
- Chip enable (CE) don't-care devices only
- Single-level cell (SLC) and multilevel cell (MLC) devices
- Device identification based on ONFI or ROM table
- ECC correction: 8 bits per sector for most devices (16 bits per sector for devices with large spare area)
- GPMC timings are adjusted for NAND access.
- The GPMC clock is 133 MHz.
- The device is connected to CS0.
- The gpmc_wait0 wait-pin signal is connected to the NAND BUSY output.

- Four physical blocks are searched for an image. Block size depends on the device.

For NAND memory booting, no user intervention is needed; the information in the following subsections is included for debugging. Only the CH, which is not mandatory, lets the user change clock settings and GPMC parameters. Failure in CH copying causes a return to the main booting procedure, which selects the next device for booting.

33.3.7.4.1 Initialization and NAND Detection

The initialization routine for NAND consists of three parts: GPMC initialization, device detection with parameter determination, and bad block detection.

- GPMC initialization

  The GPMC interface is configured so that it can access NAND. Because NAND memories do not need the address bus, it is released. The data bus width is initially set to 8 bits. If necessary, it is changed to 16 bits after the device parameters are determined. Table 33-41 shows the GPMC configuration used during NAND boot. Table 33-41 is included for debug information.

Table 33-41. NAND Timing Parameters

Parameter	Value (Clock Cycles)	Register Initialization (where i = 0–7)	Reset Value
Write cycle time	20	GPMC_CONFIG5_i[12:8] WRCYCLETIME = 0x14	0x11
Read cycle time	20	GPMC_CONFIG5_i[4:0] RDCYCLETIME = 0x14	0x11
CS low time	0	GPMC_CONFIG2_i[3:0] CSONTIME = 0x0	0x1
CS low to OE low time	5	GPMC_CONFIG4_i[3:0] OEONTIME = 0x5	0x6
CS low to OE high time	16	GPMC_CONFIG4_i[12:8] OEOFFTIME = 0x10	0x10
CS low to WE low time	1	GPMC_CONFIG4_i[19:16] WEONTIME = 0x1	0x5
CS low to WE high time	15	GPMC_CONFIG4_i[28:24] WEOFFTIME = 0xF	0x10
CS low to data latch time	14	GPMC_CONFIG5_i[20:16] RDACCESSTIME = 0xE	0xF

- Device detection and parameters

  The ROM code first performs an initial wait for device auto initialization (with a 250-ms time-out) with polling of the ready information. Then, it must identify the NAND type connected to the GPMC interface. The GPMC is initialized using 8 bits, asynchronous mode. The NAND device is reset (command FFh) and its status is polled until ready for operation (with a 100-ms time-out). The ONFI Read ID (command 90h/address 20h) is sent to the NAND device. If it replies with the ONFI signature (4 bytes) then a Read parameters page (command ECh) is sent. The information provided in Table 33-42 is then extracted: page size, spare area size, number of pages per block, and the addressing mode. The remaining data bytes from the parameter page stream are ignored.

Table 33-42. ONFI Parameters Page Description

Offset	Description	Size (Bytes)
6	Features supported	2
80	Number of data bytes per page	4
84	Number of spare bytes per page	2
92	Number of pages per block	4
101	Number of address cycles	1

  If the ONFI Read ID command fails (it will be the case with any device not supporting ONFI), then the device is reset again with polling for device to be ready (with 100-ms time-out). Then, the standard Read ID (command 90h/address 00h) is sent. If the Device ID (second byte of the ID byte stream) is recognized as being a supported device, then the device parameters are extracted from an internal ROM code table. Table 33-43 lists the supported NAND devices.

Table 33-43. Supported NAND Devices

Capacity	Device ID	Bus Width	Page Size in Bytes
512 Mibit	F0h	8	2048
512 Mibit	C0h	16	2048
512 Mibit	A0h	8	2048
512 Mibit	B0h	16	2048
512 Mibit	F2h	8	2048
512 Mibit	C2h	16	2048
512 Mibit	A2h	8	2048
512 Mibit	B2h	16	2048
1 Gibit	F1h	8	2048
1 Gibit	C1h	16	2048
1 Gibit	A1h	8	2048
1 Gibit	B1h	16	2048
2 Gibit	DAh	8	2048
2 Gibit	CAh	16	2048
2 Gibit	AAh	8	2048
2 Gibit	BAh	16	2048
2 Gibit	83h	8	2048
2 Gibit	93h	16	2048
4 Gibit	DCh	8	2048
4 Gibit	CCh	16	2048
4 Gibit	ACh	8	2048 (4096)
4 Gibit	BCh	16	2048 (4096)
4 Gibit	84h	8	2048
4 Gibit	94h	16	2048
8 Gibit	D3h	8	2048 (4096)
8 Gibit	C3h	16	2048 (4096)
8 Gibit	A3h	8	2048 (4096)
8 Gibit	B3h	16	2048 (4096)
8 Gibit	85h	8	2048
8 Gibit	95h	16	2048
16 Gibit	D5h	8	2048 (4096)
16 Gibit	C5h	16	2048 (4096)
16 Gibit	A5h	8	2048 (4096)
16 Gibit	B5h	16	2048 (4096)
16 Gibit	86h	8	2048
16 Gibit	96h	16	2048
32 Gibit	D7h	8	2048 (4096)
32 Gibit	C7h	16	2048 (4096)
32 Gibit	A7h	8	2048 (4096)
32 Gibit	B7h	16	2048 (4096)
32 Gibit	87h	8	2048
32 Gibit	97h	16	2048
64 Gibit	DEh	8	2048 (4096)
64 Gibit	CEh	16	2048 (4096)
64 Gibit	AEh	8	2048 (4096)
64 Gibit	BEh	16	2048 (4096)

After retrieving parameters from the table, the page size and block size are updated based on the fourth byte of the NAND ID data. Because of inconsistency among manufacturers, only devices recognized to be at least 2 Gibit have these parameters updated. Therefore, the ROM code supports 4-KiB page devices, but only if their size, according to the table, is at least 2 Gibit. Devices that are smaller than 2 Gibit have the block size parameter set to 128 KiB (when the page size is 2 KiB). Table 33-44 lists the fourth ID data byte encoding used in the ROM code.

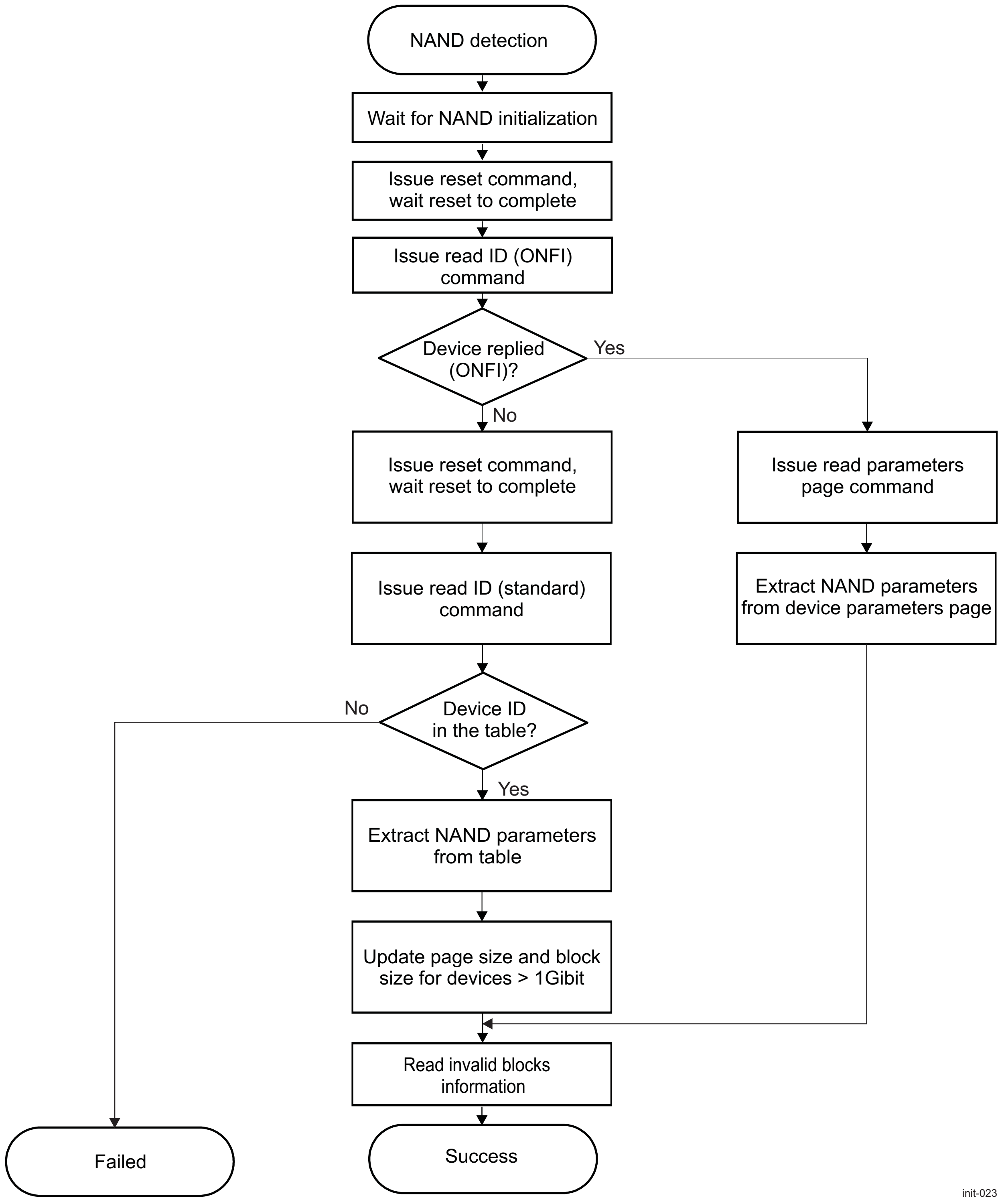
Table 33-44. Fourth NAND ID Data Byte

Item	Description	I/O Number							
		7	6	5	4	3	2	1	0
Page size	512 bytes							0	0
	2048 bytes							0	1
	4096 bytes							1	0
	8192 bytes							1	1
Cell type [(1)]	2 levels					0	0		
	4 levels					0	1		
	8 levels					1	0		
	16 levels					1	1		
Block size	64 KiB			0	0				
	128 KiB			0	1				
	256 KiB			1	0				
	512 KiB			1	1				

(1) Read by ROM code only when the manufacturer code (first ID byte) is 98h

Figure 33-18 shows the detection procedure. When the NAND device is successfully detected, the ROM code changes the GPMC to 16-bit bus width, if necessary.

Figure 33-18. NAND Device Detection

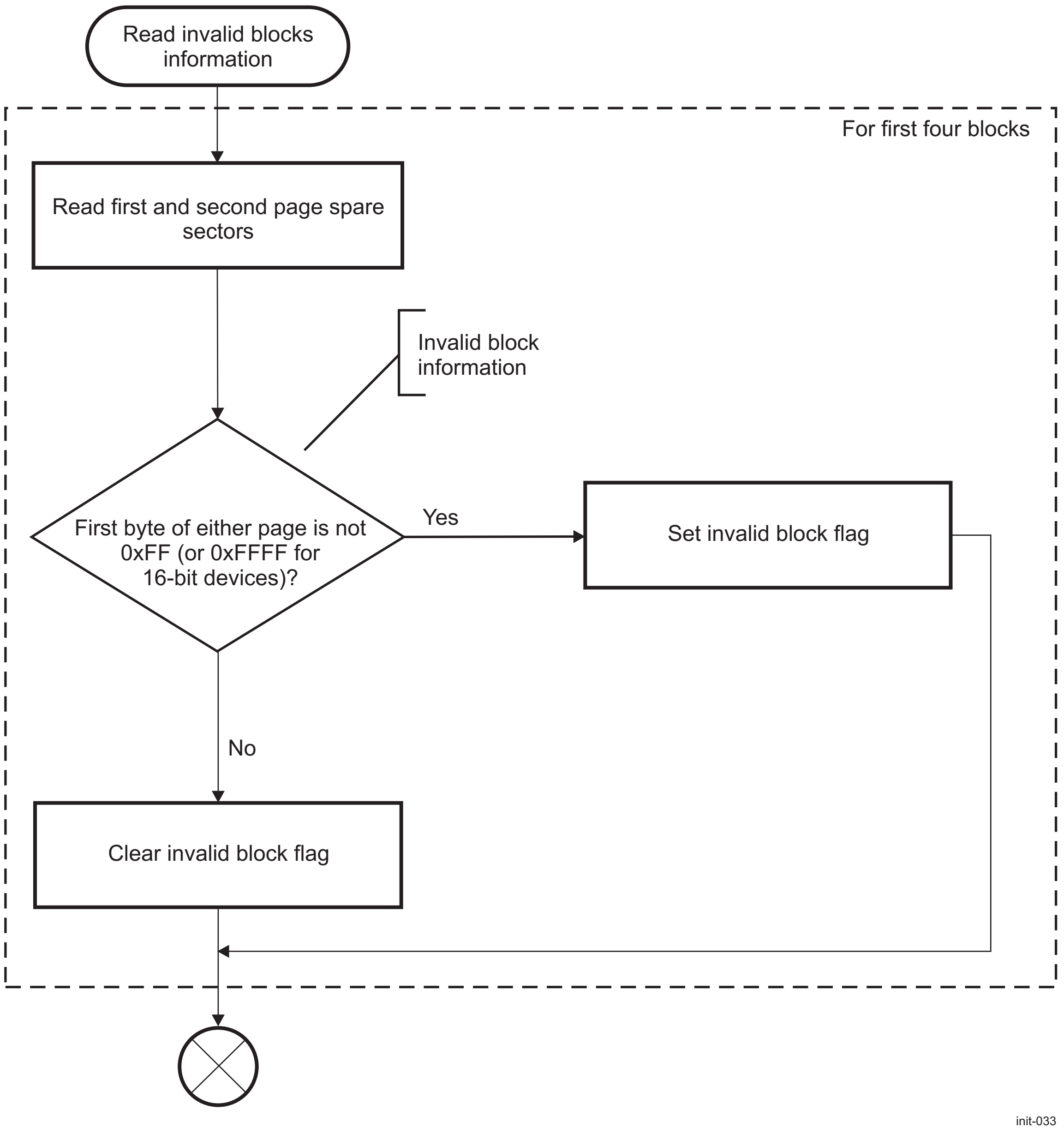

NAND detection
Wait for NAND initialization
Issue reset command, wait reset to complete
Issue read ID (ONFI) command
Device replied (ONFI)?
Yes
No
Issue reset command, wait reset to complete
Issue read parameters page command
Issue read ID (standard) command
Extract NAND parameters from device parameters page
No
Device ID in the table?
Yes
Extract NAND parameters from table
Update page size and block size for devices > 1Gibit
Read invalid blocks information
Failed
Success
init-023

- Bad block verification

  Invalid blocks contain invalid bits whose reliability cannot be ensured by the manufacturer. These bits are identified in the factory or during the programming and reported in the initial invalid block information in the spare area on the first and second page of each block. Because the ROM code looks for an image in the first four blocks, it detects the validity status of these blocks. Blocks detected as invalid are not accessed later. Block validity status is coded in the spare areas of the first two pages of a block (first byte equal to FFh in the first and second pages for an 8-bit device/first word equal to FFFFh in the first and second pages for a 16-bit device).

Figure 33-19 shows the invalid block detection routine. The routine consists in reading spare areas and checking the validity data pattern.

Figure 33-19. Bad NAND – Invalid Block Detection


Read invalid blocks information
For first four blocks
Read first and second page spare sectors
Invalid block information
First byte of either page is not 0xFF (or 0xFFFF for 16-bit devices)?
Yes
Set invalid block flag
No
Clear invalid block flag
init-033

33.3.7.4.2 NAND Read Sector Procedure

During the booting procedure, the ROM code reads 512-byte sectors from the NAND device. The reading fails in two cases:

- The accessed sector is in a block marked as invalid.
- The accessed sector contains an error that cannot be corrected with ECC.

The ROM code uses normal read (command 00h 30h) for reading NAND page data.

Page data can contain errors caused by memory alteration. The ROM code uses an ECC correction algorithm to detect and possibly correct those errors. The default ECC correction applied is BCH 8b/sector using the GPMC and ELM hardware.

For device ID codes D3h, C3h, D5h, C5h, D7h, C7h, DEh, and CEh when the manufacturer code (first ID byte) is 98h, the cell type information is checked in the fourth byte of ID data. If it is equal to 10b, the ECC correction applied is BCH 16b/sector.

The BCH data is automatically calculated by the GPMC on reading each 512-byte sector. The computed ECC is compared against the ECC stored in the spare area for the corresponding page. Depending on the page size, the amount of ECC data bytes stored in the corresponding spare area is different. Figure 33-20 and Figure 33-21 show the mapping of ECC data inside the spare area for 2-KiB page and 4-KiB page devices, respectively. If both ECC data are equal, the read sector function returns the read 512-byte sector without error. Otherwise, the ROM code tries to correct errors in the corresponding sector (this procedure is assisted by the ELM hardware) and returns the data if successful. If errors are uncorrectable, the function returns with FAIL.

Figure 33-20. ECC Data Mapping for 2-KiB Page and 8b BCH Encoding

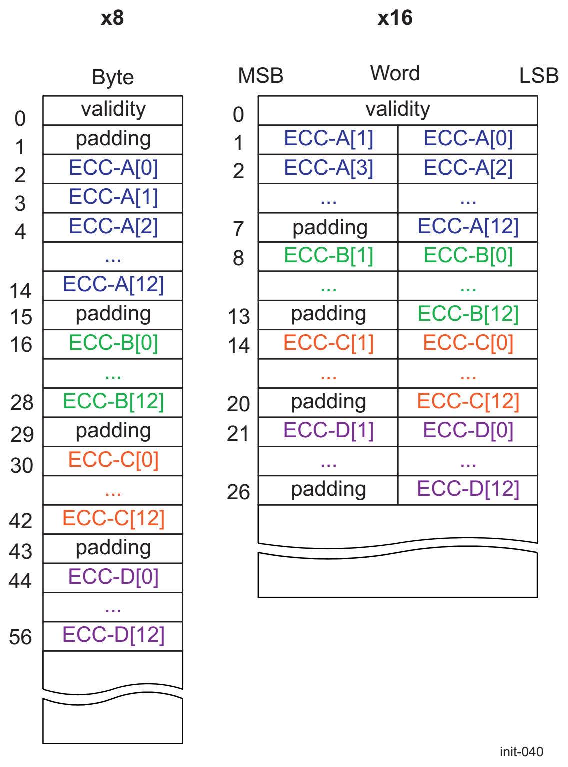
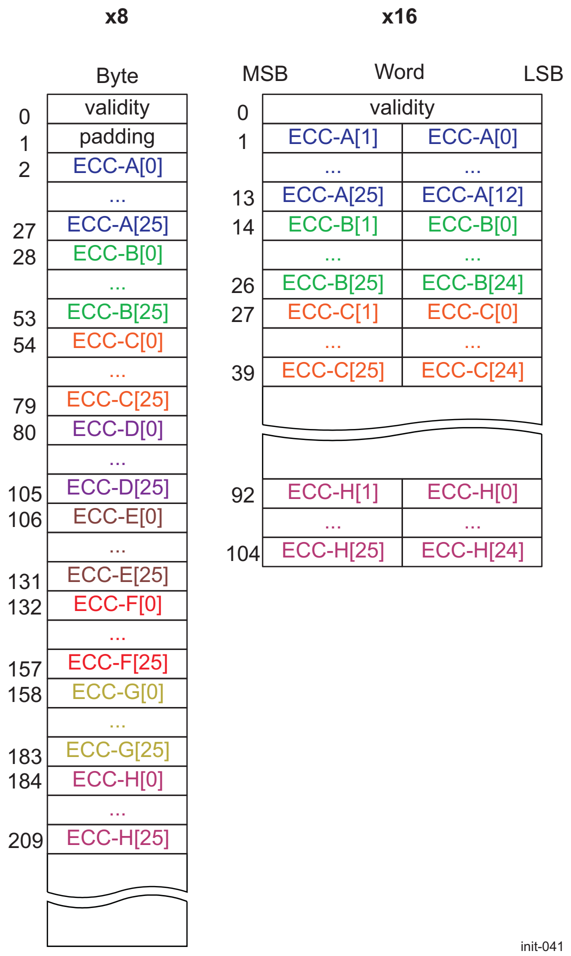


Figure 33-21. ECC Data Mapping for 4-KiB Page and 16b BCH Encoding



33.3.7.5 SPI/QSPI Flash Devices

SPI/QSPI Flash memories provide a storage solution for systems with limited space, pins and power.

The ROM code support for SPI/QSPI devices has the following characteristics:

- 24-bit addressing, up to 128 Mbit (16 MiB), no banking
- QSPI1 on CS0 is the communication interface
- Uses Mode 3:
  - Clock inactive state = high
  - Data input captured on rising edge of clock
  - Data output generated on falling edge of clock
- QSPI 4-bit data read mode at 48 MHz in configuration port mode
  - Read command is 0x6B (Fast Quad Read), 3 address bytes,1 dummy bytes and read type is quad read
  - ROM will not perform any quad-enable sequence nor bank register update
- SPI 1-bit data read mode at 12 MHz in configuration port mode
  - Read command is 0x03 (Single Read), 3 address bytes, 0 dummy bytes and read type is normal read.
- Up to four redundant images can be stored on SPI/QSPI flash. The offset between them is set in sysboot[7:6] as described in Table 33-8. If not using the redundant SBL feature, the offset set in sysboot[7:6] is a don't care. A booting image is considered to be present, when the first 4 bytes word is

not equal to 0000 0000h or FFFF FFFFh.

NOTE: ROM code does not perform any specific action to detect, reset or power up the QSPI device. QSPI is assumed to be properly powered and reset to be completed before every attempt to boot by ROM code.

33.3.7.6 eMMC Memories and SD Cards

The device allows booting from eMMC embedded memories or SD cards connected to device embedded MMC2 or MMC1 controllers I/Os, respectively. The booting interface is selected by configuration of the SYSBOOT pins.

The device high speed MMC/SD/SDIO host controller handles the physical layer, while the ROM code handles the simplified logical protocol layer (read-only protocol). A limited range of commands is implemented in the ROM code.

The SD card interface supports 1.8-V/3-V digital I/Os. The selection on I/O voltage level is done using the PBIAS circuitry. The PBIAS reference level is sensed by the ROM code and PBIAS cell voltage setup appropriately before any communication with the SD card device. The eMMC interface supports also 1.8-V/3-V digital I/Os but with the difference that there is no PBIAS circuitry to select the I/O voltage level. This voltage depends only on the voltage provided to the eMMC associated power pads.

The device supports three booting modes:

- Raw (Boot): The booting image is read from one of the selectable partitions in raw mode (this mode is also called Alternative Boot Operation mode and applies to eMMC only).
- Raw (UDA): The booting image is read from the user data area (eMMC and SD).
- File system (FAT12/16/32 with or without master boot record): The image data is read from a booting file within a file system in the user data area (eMMC and SD).

NOTE: File system mode is only supported when booting in the eMMC user area. It is not supported in the eMMC boot area.

33.3.7.6.1 eMMC Memories

The ROM code supports booting from eMMC memories, with the following conditions:

- eMMC memory devices compliant with *Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write Boot, and Sleep Modes, Dual Data Rate, Multiple Partitions Supports and Security Enhancement v4.5* from the MMCA Technical Committee. The exception is the hardware reset feature. For example, if the user software requires eMMC device hardware reset, it can be accomplished with a GPIO. To correctly boot from eMMC when using Alternative Boot Operation mode, it is recommended to tie the eMMC device RST_N signal to the platform warm reset.
- eMMC device connected to the device MMC2 controller I/O interface (only one device can be connected to the bus).
- The eMMC memory device is powered externally by a PMIC or other power supply.
- When booting from user area: Initial (default) 1-bit SDR mode, optional 4-bit and 8-bit SDR- and DDR-modes using Configuration Header
- Initial SDR- and DDR- modes supported
- Clock frequency when booting from user area:
  - Identification mode: 400 kHz
  - Data transfer mode: 10 MHz, optionally up to 48 MHz by Configuration Header
  - Support for eMMC boot partitions. eMMC booting from boot partitions (BPs), which is typical for booting in Alternative Boot operation mode, is always done at 8-bit, 48 MHz, DDR.

33.3.7.6.1.1 System Conditions and Limitations

The ROM code does not provide a bus direction control signaling in case of using MMC interface 2 with level shifters.

NOTE: The ROM Code does not support large 4-KiB sectors as defined in the latest standard.

The ROM code expects that the eMMC device is powered externally and supplies are set and stable when entering the booting procedure.

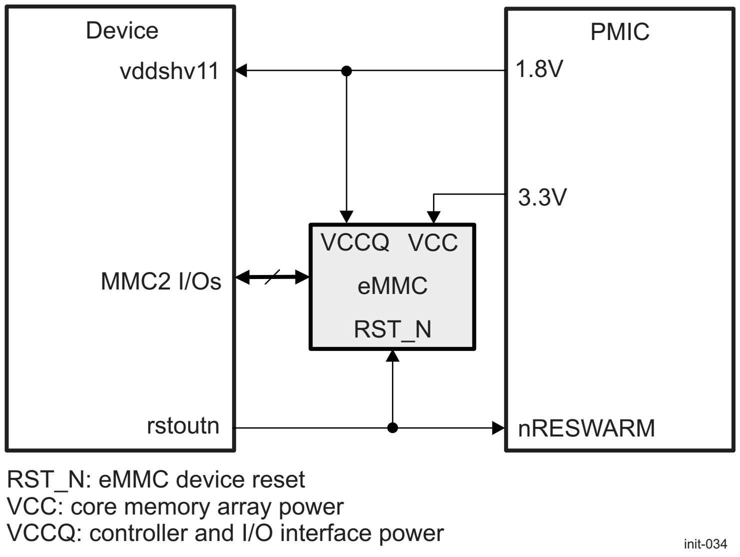
The ROM code supports the alternative boot operation mode specific to eMMC devices as described in the *Partition Management* and *Boot Operation Mode* sections of the eMMC Standard. It does not support the hardware boot operation mode (CMD line held low for 74 emmc_clk cycles) described in the same document.

33.3.7.6.1.2 eMMC Memory Connection

An eMMC device typically requires two supplies: one for the core memory array (VCC) and one for the device interface I/Os and controller (typical VCCQ = 1.8 V). Both memory device supply pins can possibly be merged into one, thus requiring only one power supply.

Figure 33-22 shows an example of the system connection between the PMIC, memory device, and device.

Figure 33-22. eMMC Connection



NOTE: For correct handling of eMMC boot partitions when using alternative boot operation mode, it is recommended to tie the eMMC RST_N signal to the signal indicating a platform warm reset (nRESWARM).

The ROM code performs the necessary I/O pin muxing configuration to route the MMC2 I/O signals on the eMMC pads depending on the selected SYSBOOT configuration.

33.3.7.6.2 SD Cards

The ROM code supports booting from SD cards under the following conditions:

- SD cards compliant with *SD Specifications Part 1 Physical Layer Specification Version 4.00*and the *SD Specifications Part 2 File System Specification Version 3.00* from the SD Association. These include low- (SDSC) and high-capacity (SDHC) cards.
- SD card connected to MMC1 controller I/Os. Only one card is allowed to be connected to the bus.

- 3-V VCC power supply
- 3-V or 1.8-V I/O voltages
- Initial 1-bit MMC mode, optional 4-bit mode
- Clock frequency:
  - Identification mode: 400 kHz
  - Data transfer mode: 10 MHz (optionally up to 19.2 MHz by Configuration Header)

33.3.7.6.2.1 System Conditions and Limitations

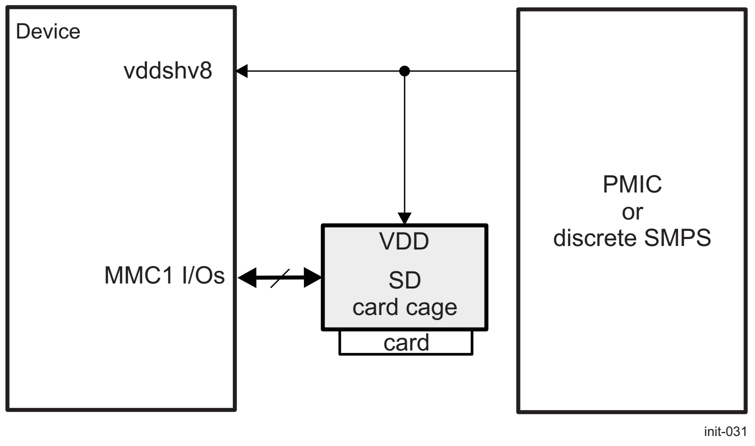
Even though the ROM Code identities SDXC cards, it does not handle its specificities such as exFAT support. It does neither support UHS-I nor UHS-II speed grades.

33.3.7.6.2.2 SD Card Connection

An SD card can be connected to the SD card interface, typically through a card cage.

Figure 33-23 shows the typical connection between the power IC, the card, and the device.

Figure 33-23. MMC/SD Card Connection



NOTE: The ROM code does not handle the card detection feature on the card cage.

33.3.7.6.2.3 Booting Procedure

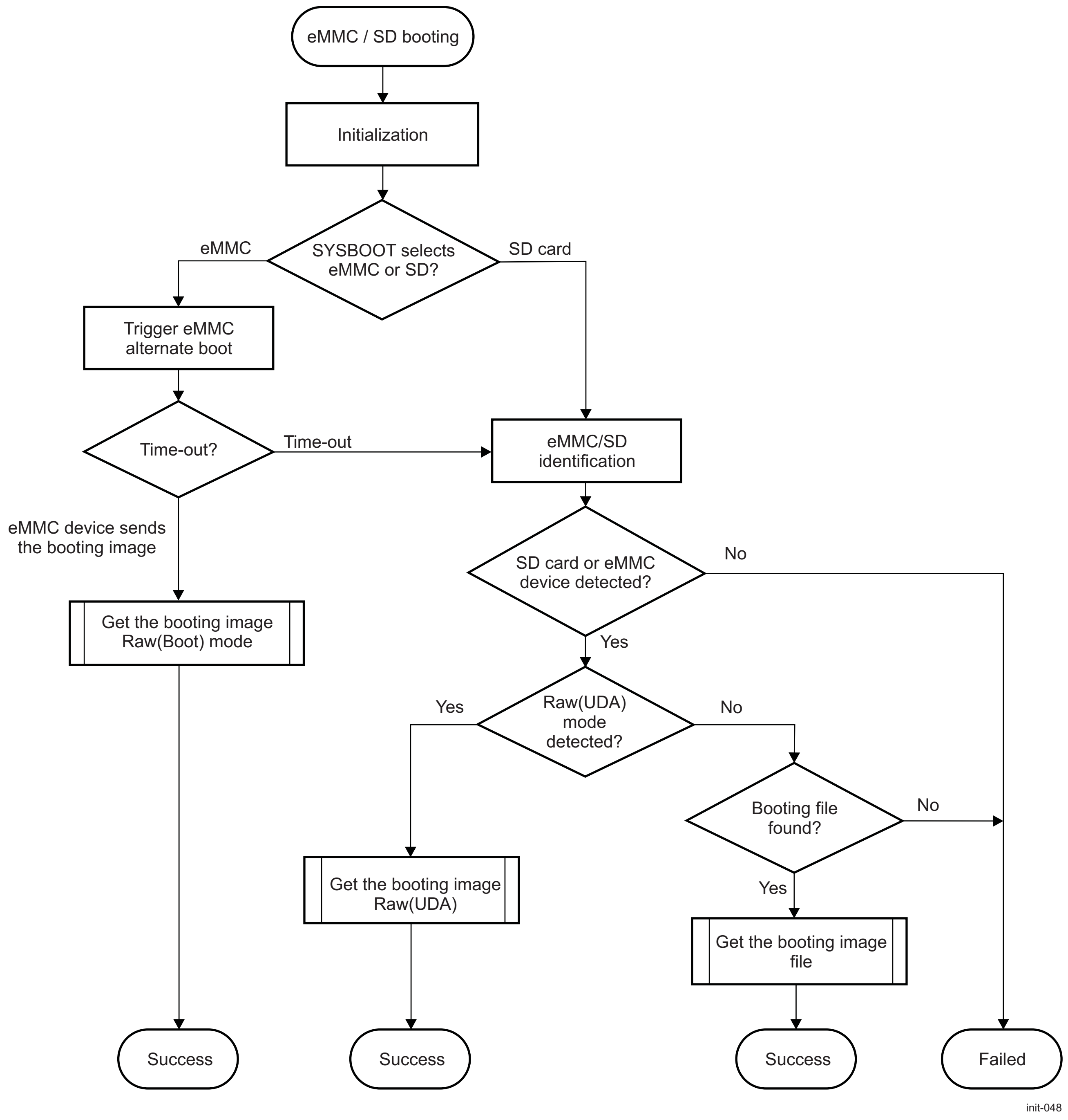
If the selected booting device is eMMC, then a first attempt to trigger alternative boot operation mode is tried. If the eMMC device replies within a fixed timeout, then the booting image is directly retrieved from one of the possible partitions specified in the local EXT_CSD.BOOT_PARTITION_ENABLE bit field for the eMMC device (this is referred to as the Raw(Boot) mode). Raw(Boot) mode improves the boot time by avoiding the regular eMMC/SD identification phase (usually done at low speed). When the booting image is retrieved, the ROM code continues the boot process.

If the time-out elapses, then the ROM code proceeds with normal identification. SD cards do not support Alternative Boot Operation mode, and therefore are always going through normal identification.

If normal identification succeeds, then it next determines whether the SD card or eMMC device contains a known file system. If the file system is known, then the booting image is extracted from the file system hierarchy. If a file system is not detected, then the Raw(UDA) mode is assumed.

Figure 33-24 is the high level flow chart of the eMMC and SD booting procedure.

Figure 33-24. eMMC and SD Booting


eMMC / SD booting
Initialization
SYSBOOT selects eMMC or SD?
eMMC
SD card
Trigger eMMC alternate boot
Time-out?
Time-out
eMMC/SD identification
eMMC device sends the booting image
SD card or eMMC device detected?
No
Yes
Get the booting image Raw(Boot) mode
Raw(UDA) mode detected?
Yes
No
Booting file found?
No
Yes
Get the booting image Raw(UDA)
Get the booting image file
Success
Success
Success
Failed
init-048

33.3.7.6.2.4 eMMC Partitions Handling in Alternative Boot Operation Mode

To trigger alternative boot operation mode, the ROM code sends a specific CMD0 + Argument 0xFFFF FFFA. Then, the ROM code waits up to 50 ms for the eMMC device to return a boot-acknowledge signal. If the time-out elapses, then the ROM code skips alternative boot operation mode. Booting from eMMC partitions can be done in 8-bit mode at 48 MHz, DDR. This configuration is static and cannot be changed.

33.3.7.6.2.4.1 eMMC Devices Preflashing

For correct handling of eMMC partitions in alternative boot operation mode, it is necessary to prepare the eMMC device at flashing time with the following settings applied to the device EXT_CSD register:

- The BOOT_ACK and BOOT_PARTITION_ENABLE fields must be updated accordingly (EXT_CSD[179], bit 6 and bits [5:3], respectively) to activate the boot acknowledge signal and select the partition from which to boot. There are several boot options selectable in the BOOT_PARTITION_ENABLE bit field, as follows:
  - Booting from boot partition 1 (BP1)
  - Booting from boot partition 2 (BP2)
  - Booting from User Area

  For more details, see the eMMC Standard documentation.
- RST_N must be enabled for correct handling of the warm reset cases (EXT_CSD[162] bits [1:0] = 0x1).
- The BOOT_BUS_WIDTH fields (EXT_CSD[177]) must be updated properly based on device alternate boot operation: 8-bit, 48 MHz, DDR mode.
- Optionally the BOOT_CONFIG_PROT (EXT_CSD[178]) may be updated for altering access permissions to the selected partitions.

NOTE: Although an alternate boot from user area is possible, alternate booting from BP1 or BP2 is recommended.

NOTE: It is possible to permanently or temporarily lock the boot configuration through the use of the BOOT_CONFIG_PROT register. For more details, see the eMMC Standard documentation.

NOTE:

- It is highly recommended to refer to the eMMC Standard for details on EXT_CSD handling.
- The EXT_CSD is updated by using a SWITCH command as detailed in the eMMC standard.
- The mentioned EXT_CSD fields are retained after a power cycle so only one “flashing” phase is required

33.3.7.6.2.4.2 eMMC Device State After ROM Code Execution

If alternative boot operation mode is successful, and the booting image is properly retrieved, then the eMMC device state remains in the BOOT state even after the execution has passed to the initial software. The initial software must bring the device out of the BOOT state.

33.3.7.6.2.4.3 Consideration on device Global Warm Reset

Once the system has booted and upon triggering a warm reset at device level (it can be triggered by the warm-reset push button, a timer, watchdog, etc.), it is important that the eMMC device is properly brought to its PRE-IDLE state so that the next Alternative Boot operation succeeds. This is ensured by the eMMC RST_N pin to be connected to device rstoutn pin (warm reset). Not doing so would make the boot procedure fail after a warm reset.

33.3.7.6.2.4.4 Booting Image Size

In Raw(Boot) mode, the size of the booting image is determined after the first sector read access. The boot image size is contained within the GP header. This ensures that the ROM code is only retrieving the necessary size of data and not the full contents of the partition.

33.3.7.6.2.4.5 Booting Image Layout

In Raw(Boot) mode only one copy of the booting image is maintained, as opposed to up to four copies in Raw(UDA) mode.

33.3.7.6.3 Initialization and Detection

If the eMMC Alternate Boot operation fails in time-out, or the requested boot device is an SD card, the ROM code initializes the memory device or card connected on the eMMC and SD card interface using 1.8 V for eMMC and standard high-voltage range (3.0 V) for SD card I/Os. If neither a card nor memory device is detected, the ROM code moves to the next booting device. The standard identification process and relative card address (RCA) assignment are performed. The ROM code assumes that only one memory or card is connected on the bus. This is done using the CMD line common to the SD and eMMC memory devices. The eMMC and SD standards describe this phase as the initialization phase. They differ in the commands involved, as described in Figure 33-26. The ROM code uses CMD55 to discriminate between eMMC and SD cards; that is, CMD55 is only supported by the SD standard. Depending on response received or not, ACMD41 (the combination of CMD55 and CMD41 for SD) or CMD1 (for MMC) is sent. If no response is received this time, no devices are connected and the ROM code exits eMMC/SD booting with FAIL.

Figure 33-25 shows the eMMC/SD detection procedure.

Figure 33-25. SD/eMMC Detection Procedure (part 1)

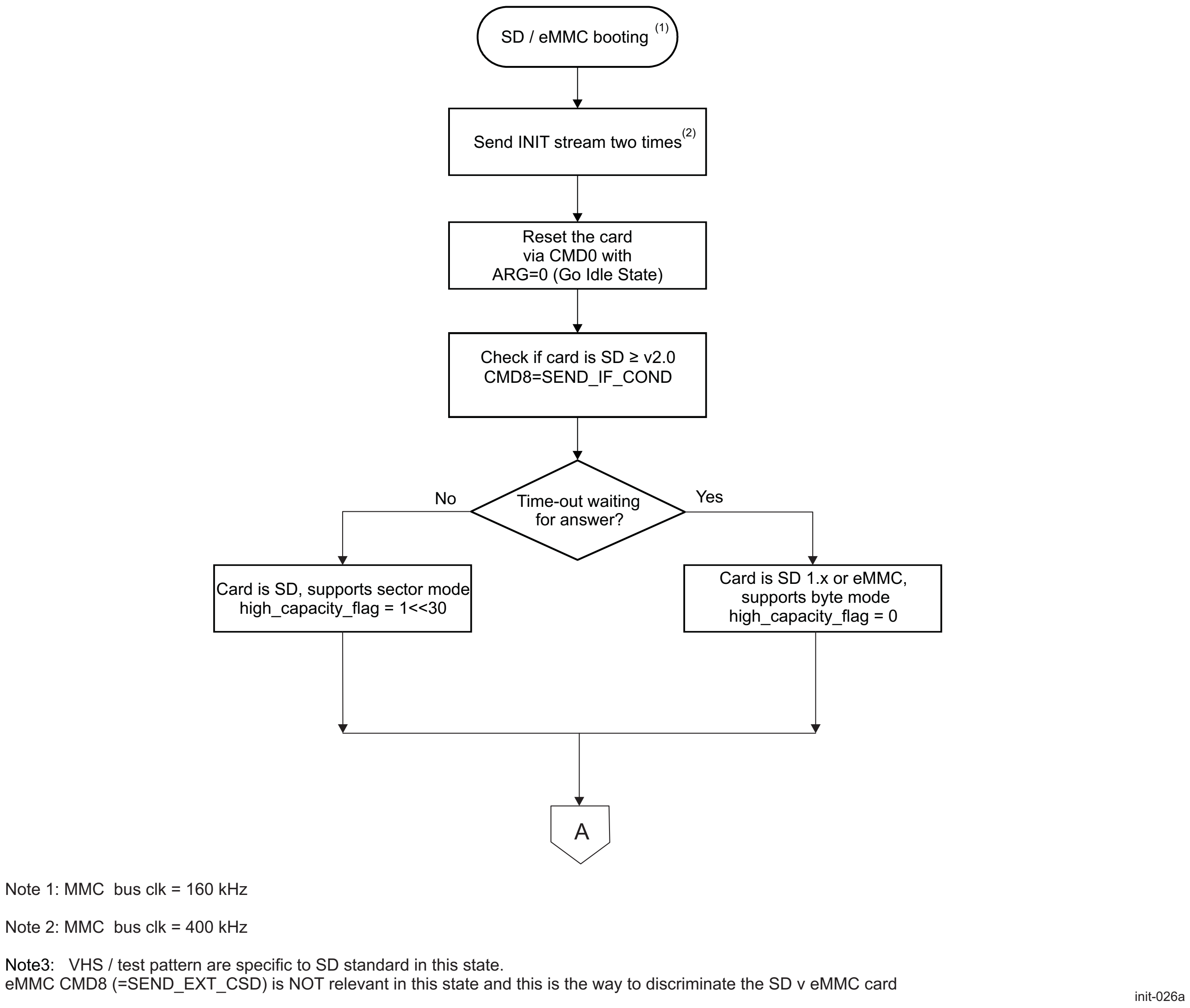
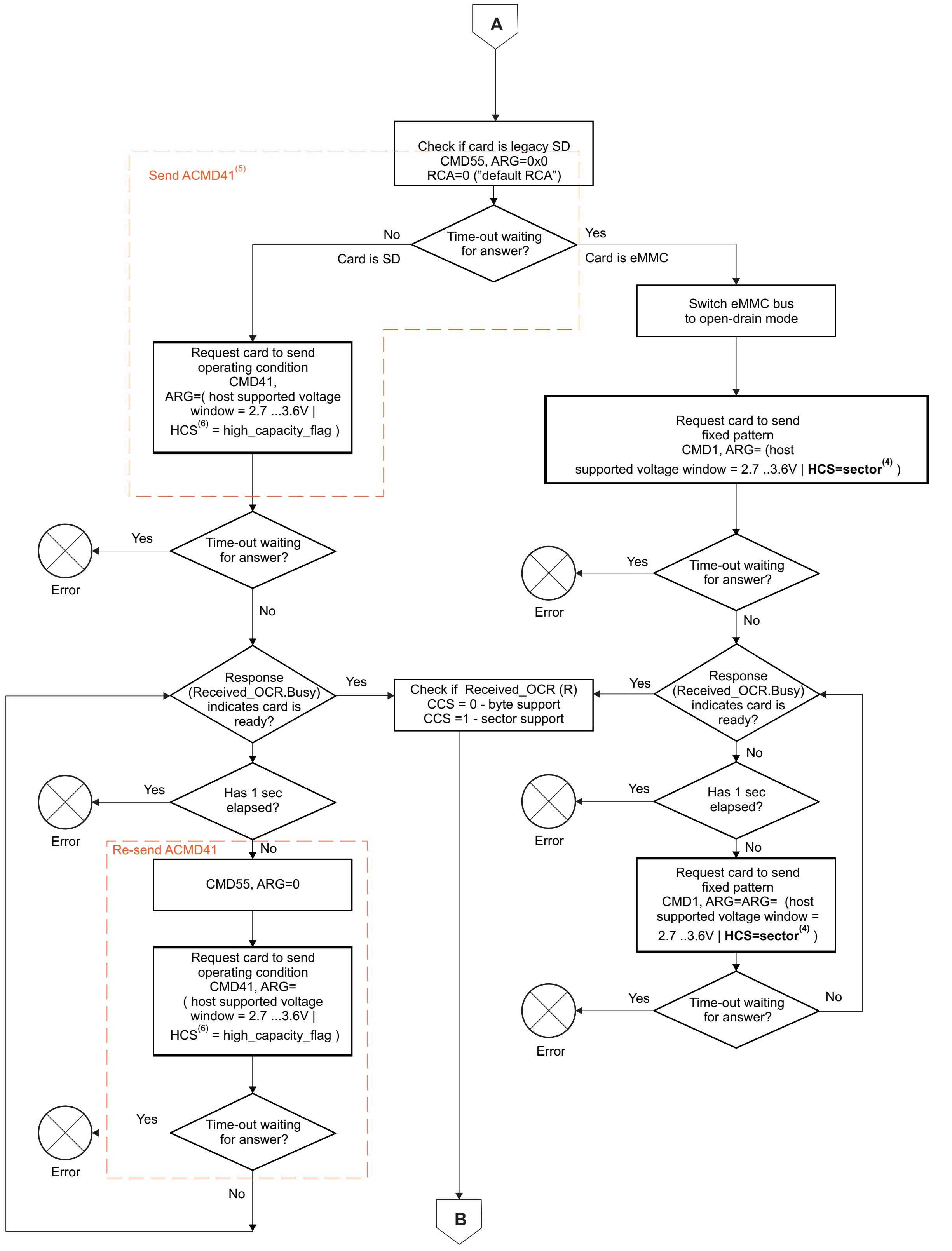

SD / eMMC booting (1)
Send INIT stream two times (2)
Reset the card via CMD0 with ARG=0 (Go Idle State)
Check if card is SD ≥ v2.0 CMD8=SEND_IF_COND
Time-out waiting for answer?
No
Yes
Card is SD, supports sector mode high_capacity_flag = 1<<30
Card is SD 1.x or eMMC, supports byte mode high_capacity_flag = 0
A
Note 1: MMC bus clk = 160 kHz
Note 2: MMC bus clk = 400 kHz
Note3: VHS / test pattern are specific to SD standard in this state.
eMMC CMD8 (=SEND_EXT_CSD) is NOT relevant in this state and this is the way to discriminate the SD v eMMC card
init-026a

Figure 33-26 shows the eMMC/SD detection procedure.

Figure 33-26. SD/eMMC Detection Procedure (part 2)

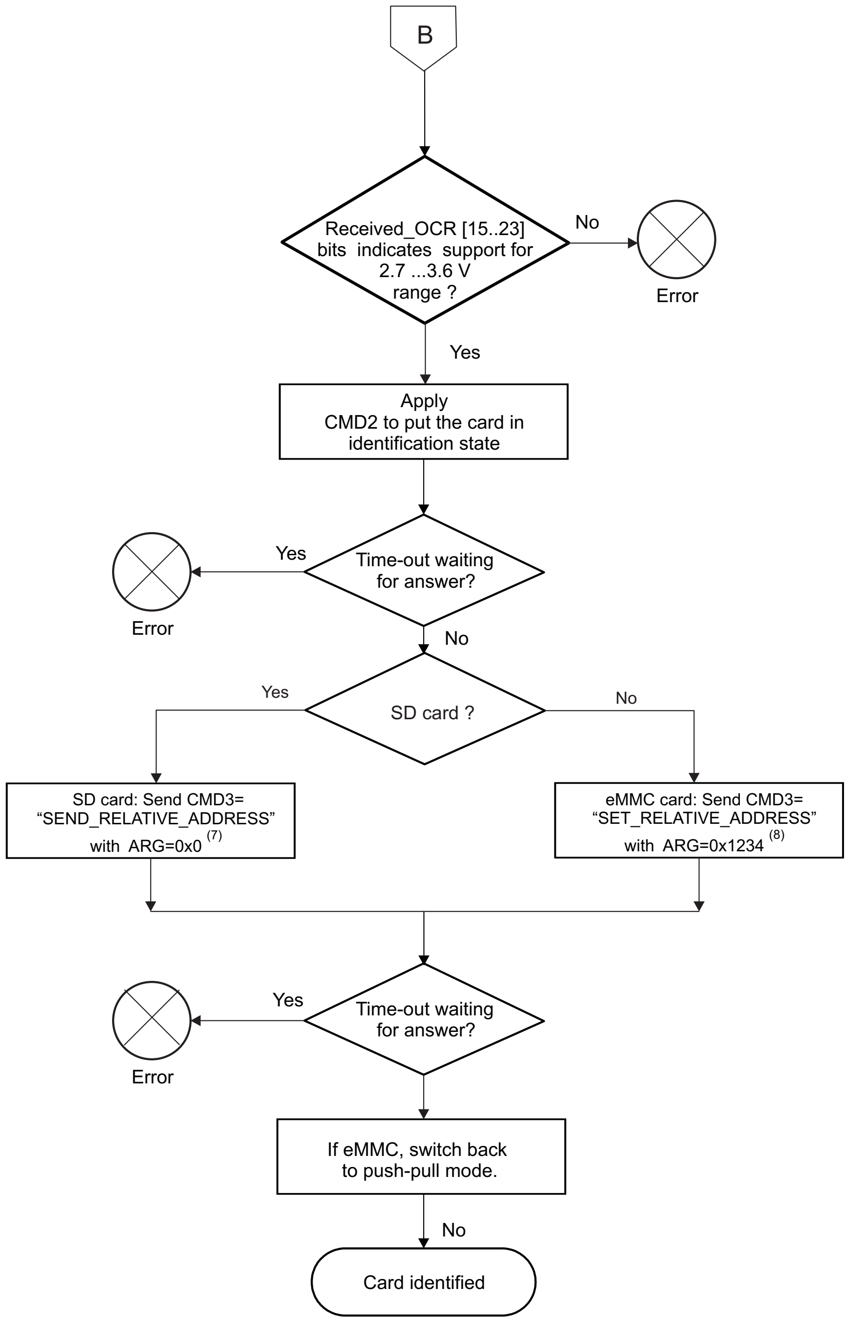

A
Check if card is legacy SD
CMD55, ARG=0x0
RCA=0 ("default RCA")
Send ACMD41(5)
Time-out waiting for answer?
No
Card is SD
Yes
Card is eMMC
Switch eMMC bus to open-drain mode
Request card to send operating condition CMD41, ARG=(host supported voltage window = 2.7 ...3.6V | HCS(6) = high_capacity_flag)
Request card to send fixed pattern CMD1, ARG= (host supported voltage window = 2.7 ..3.6V | HCS=sector(4))
Time-out waiting for answer?
Yes
Error
No
Response (Received_OCR.Busy) indicates card is ready?
Check if Received_OCR (R) CCS = 0 - byte support CCS =1 - sector support
Has 1 sec elapsed?
Re-send ACMD41
CMD55, ARG=0
Request card to send operating condition CMD41, ARG= (host supported voltage window = 2.7 ...3.6V | HCS(6) = high_capacity_flag)
Request card to send fixed pattern CMD1, ARG=ARG= (host supported voltage window = 2.7 ..3.6V | HCS=sector(4))
B

Note 4: Host indicates it is capable of sector addressing
Note 5: Refered to as the first ACMD41 that starts initialization in SD standard. Subsequent ACMD41 provide same argument
Note 6 : Host sets hcs per response from CMD8 as described in SD standard

init-026b

Figure 33-27 shows the eMMC/SD detection procedure.

Figure 33-27. SD/eMMC Detection Procedure (part 3)


B
Received_OCR [15..23] bits indicates support for 2.7 ...3.6 V range ?
No
Error
Yes
Apply CMD2 to put the card in identification state
Time-out waiting for answer?
Yes
Error
No
SD card ?
Yes
No
SD card: Send CMD3= "SEND_RELATIVE_ADDRESS" with ARG=0x0 (7)
eMMC card: Send CMD3= "SET_RELATIVE_ADDRESS" with ARG=0x1234 (8)
Time-out waiting for answer?
Yes
Error
If eMMC, switch back to push-pull mode.
No
Card identified

Note 7 : Card sends back a new RCA value that ROM Code keeps internally
Note 8: Card stores the new RCA value provided by the ROM Code.

init-026c

33.3.7.6.4 Read Sector Procedure

The contents of an eMMC or SD card may be formatted as raw binary (referred to as Raw(UDA) or within a FAT file system. The ROM Code reads out sectors from raw image or the booting file within the file system and boots from it.

- **Raw mode**

In Raw(UDA) mode, an image can be at one of the four consecutive locations in the main area: offset 0x0 (0 KiB)/0x20000 (128 KiB)/0x40000 (256 KiB)/0x60000 (384 KiB). For this reason, the size of a booting image must not exceed 128 KiB. However, a device with an image greater than 128 KiB can be flash starting at one of the aforementioned locations. Therefore, the ROM code does not check the image size. The only drawback is that the image crosses the subsequent image boundary. Raw mode is detected by reading sectors 0, 256, 512, and 768. The content of these sectors is verified for the presence of a TOC structure. GP header must be located at the beginning of the booting image, as described in Section 33.3.8.2, *Configuration Header*. Image data is read directly from continuous sectors of a card. If raw mode is not detected, file system mode is assumed.

- File system handling

  The read sector procedure uses the standard SD/eMMC read data procedure. The sector address is generated based on the booting memory file map collected during initialization. Thus, the ROM code can freely address sectors in the booting file space.

33.3.7.6.5 File System Handling

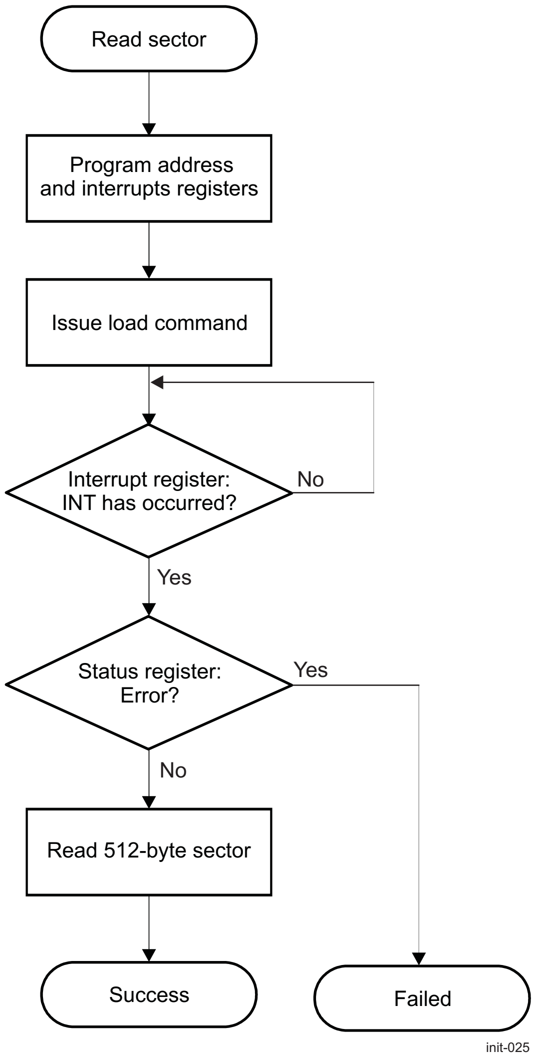
The eMMC/SD cards can hold a file system that the ROM code reads. The image used by the booting procedure is taken from a booting file named MLO. This file must be in the root directory on an ***active*** primary partition of type FAT12/16 or FAT32.

An eMMC/SD card can be configured as floppy-like or hard-drive-like:

- When acting like a floppy, the content of the card is a single FAT12/16/32 file system without an MBR holding a partition table.
- When acting like a hard drive, an MBR is present in the first sector of the card. This MBR holds a table of partitions, one of which must be FAT12/16/32, primary, and active.

According to the *MultiMediaCard FAT16 File System Specification* from the MMCA Technical Committee, the card must always hold an MBR, except when using a floppy-like file system. However, depending on the operating system used, the eMMC/SD card is formatted with or without partitions (using an MBR). The ROM code supports both types: floppy-like or hard-drive-like. The ROM code retrieves a map of the booting file from the FAT. The booting file map is a collection of all FAT entries related to the booting file (a FAT entry points to a cluster holding part of the file). The booting procedure uses this map to access any 512-byte sector in the booting file without involving the ROM code FAT module. Figure 33-28 shows the complete process.

Figure 33-28. SD/MMC Get Booting File



33.3.7.6.5.1 MBR and FAT File System

This paragraph describes functions used by the ROM code to recognize whether an MBR with a FAT is used. It is not intended to fully describe the MBR and the FAT file system detection and reading procedure. The ROM code can detect FAT12/16/32 allocation table types. It cannot boot on devices with NTFS or Linux® FS partitions. Some memory devices that support file systems can be formatted with or without MBR; therefore, the first task of the ROM code is to detect whether the device is holding an MBR in the first sector.

The MBR is the first sector of a memory device. It consists of executable code, four partition entries, and one signature. The aim of such a structure is to divide the hard disk in partitions used primarily to boot different systems (for instance, Microsoft Windows®). Table 33-45 describes this structure, and Table 33-46 describes the partition table entry.

Table 33-45. Master Boot Record Structure

Offset	Length (Bytes)	Entry Description
0000h	446	Optional code
01BEh	16	Partition table entry
01CEh	16	Partition table entry
01DEh	16	Partition table entry

Table 33-45. Master Boot Record Structure (continued)

Offset	Length (Bytes)	Entry Description
01EEh	16	Partition table entry
01FEh	2	Signature (0xAA55)

Table 33-46. Partition Table Entry

Offset	Length (Bytes)	Entry Description	Value
0000h	1	Partition state	00h: Inactive 80h: Active
0001h	1	Partition start head	Hs
0002h	2	Partition start cylinder and sector	Cs[7:0] – Cs[9:8] – Ss[5:0]
0004h	1	Partition type	01h: FAT12 04h, 06h, 0Eh: FAT16 0Bh, 0Ch, 0Fh: FAT32
0005h	1	Partition end head	He
0006h	2	Partition end cylinder and sector	Ce[7:0]–Ce[9:8]–Se[5:0]
0008h	4	First sector position relative to the beginning of media	LBAs = Cs.H.S + Hs.S + Ss – 1
000Ch	4	Number of sectors in partition	LBAe = Ce.H.S + He.S + Se – 1 Nb s = LBAe – LBAs + 1

SD/eMMC booting consists of the following steps:

1. Detection of MBR

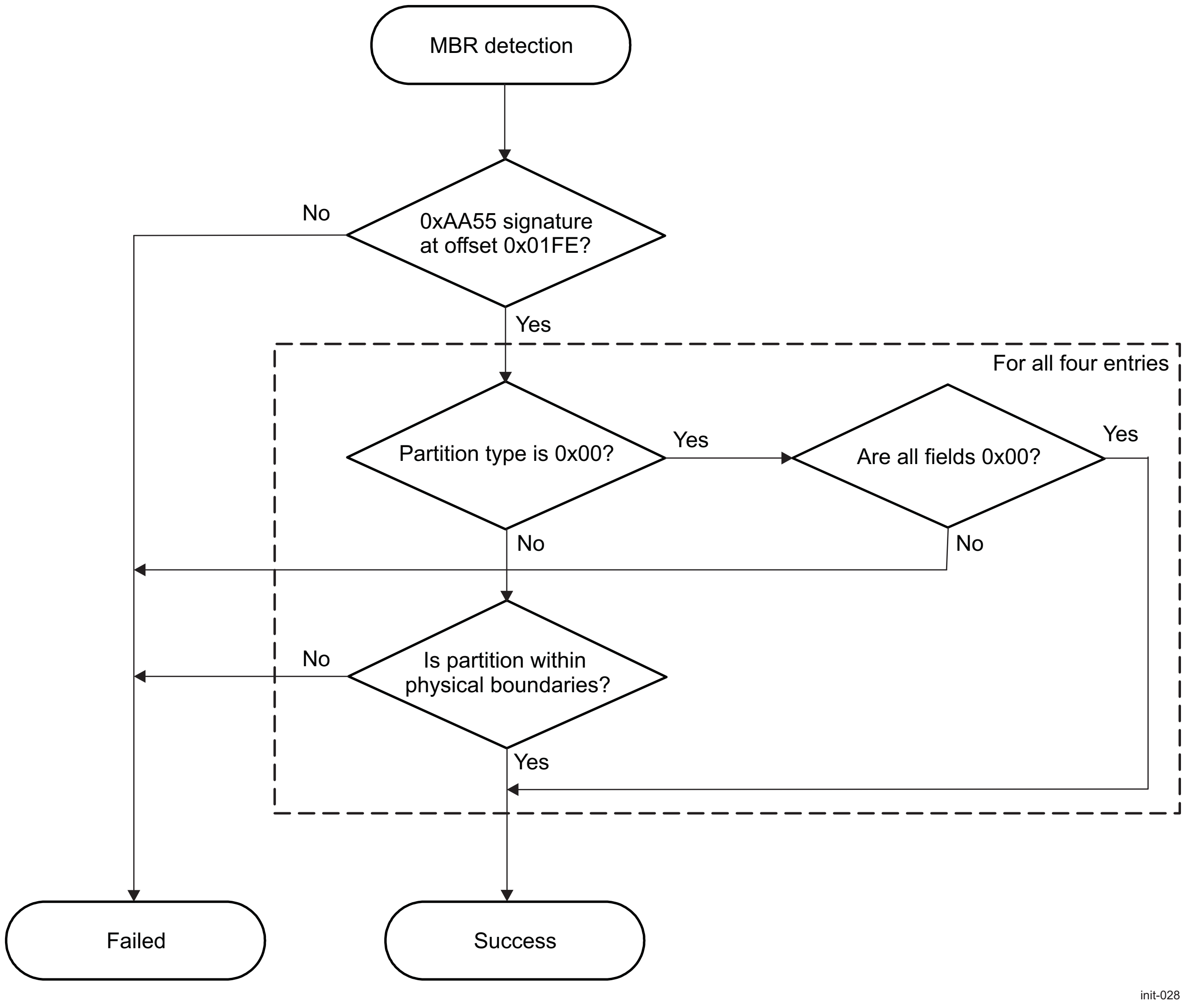
   The ROM code first checks whether the MBR signature is present, and then it searches an active FAT12/16/32 partition in all four MBR partition entries, based on the Type field. If the MBR entries are not valid, or if no usable partition is found, the ROM code returns to the booting procedure with FAIL. The extended partitions are not checked; the booting file must reside in a primary partition. Each partition entry is checked to determine the following:

   (a) If the partition type is set to 00h, all fields in the entry must be 00h.

   (b) The partition is within physical boundaries (that is, the partition is inside and it fits the total physical sectors).

   See Figure 33-29 for more information about MBR detection.

Figure 33-29. MBR Detection Procedure



2. Get the MBR partition.

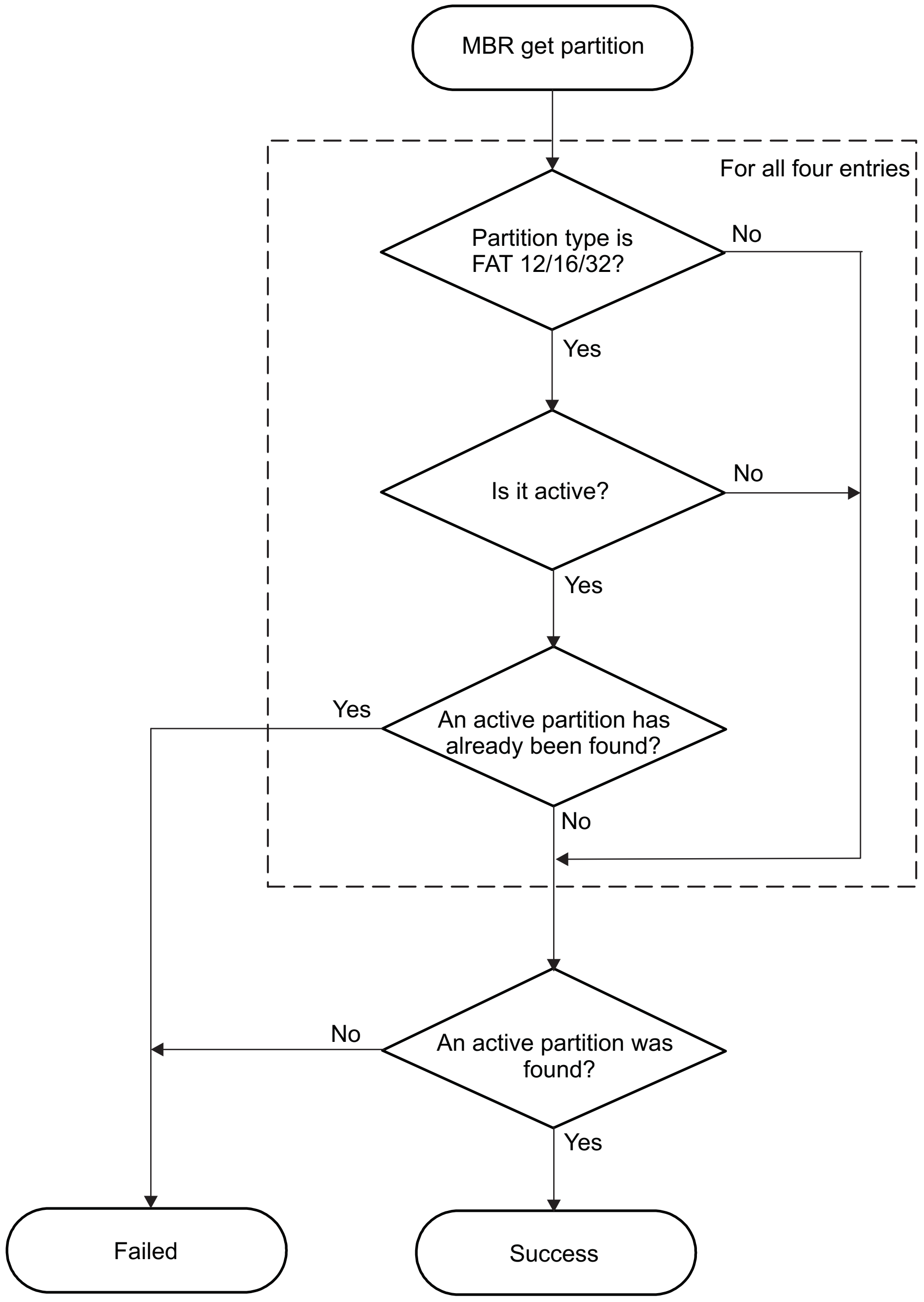
   Once identified, the ROM code gets the partition using the procedure described in Figure 33-29. The partition type is checked to be FAT12/16 or FAT32. Its state must be 00h (inactive) or 80h (active). The ROM code returns with FAIL if no active primary FAT12/16/32 is found, or the test fails if there is more than one active partition. If an active partition is found, its first sector is read and used later. If no MBR is present (in case of a floppy-like system), the first sector of the device is read and used later. The read sector is checked to be a valid FAT12/16 or FAT32 partition. If this fails, the ROM code returns with FAIL if another partition type is used (for instance, Linux FS) or if the partition is not valid.

   The FAT file system consists of several parts:

   - Boot sector, which holds the BIOS parameter block (BPB). Not all are used by the ROM code.
   - FAT, which describes the use of each cluster of the partition
   - Data area, which holds the files, directories, and root directory (for FAT12/16, the root directory has a specific fixed location)

   To check whether a sector holds a valid FAT12/16/32 partition, many fields of the boot sector (used by all FAT types) that require specific values are checked. Figure 33-30 shows more about getting the partition.

Figure 33-30. MBR, Get Partition


MBR get partition
For all four entries
Partition type is FAT 12/16/32?
No
Yes
Is it active?
No
Yes
An active partition has already been found?
Yes
No
An active partition was found?
No
Yes
Failed
Success
init-029

3. Find the booting file.

   When a partition is found, the root directory entries are searched for a booting file named MLO in the root directory of the FAT12/16/32 file system. The file is not searched in any other location. For a FAT12/16 file system, the root directory has a fixed location, which is cluster 0. For a FAT32 file system, its cluster location is given by BPB_RootClus. The formula to find the sector number (relative to device sector 0, not partition sector 0) of a cluster is given by the following equation:

$$Cluster_{sector} = BPB\_HiddSec + BPB\_RsvdSecCnt + BPB\_NumFATs \cdot BPB\_FATSz + Cluster \cdot BPB\_SecPerCLus$$

init-E001

NOTE: BPB_FATSz is BPB_FATSz16 for FAT12/16, or BPB_FATSz32 for FAT32.

NOTE: The BPB_HiddSec field can contain 0, even though the FAT file system is somewhere other than on sector 0 (floppy-like). The ROM code uses the partition offset taken from the MBR instead of this field, which can be wrong. If no MBR is found (floppy-like), the value 0 is used.

Each entry in the root directory is 32 bytes and holds information about the file (the filename, date of creation, rights, cluster location, and so forth). See Table 33-47.

Table 33-47. FAT Directory Entry

Offset	Length (Bytes)	Name	Description
0000h	11	DIR_Name	Short Name (8 + 3)
000Bh	1	DIR_Attr	File attributes: 01h – ATTR_READ_ONLY 02h – ATTR_HIDDEN 04h – ATTR_SYSTEM 08h – ATTR_VOLUME_ID 10h – ATTR_DIRECTORY 20h – ATTR_ARCHIVE 0Fh – ATTR_LONG_NAME
000Ch	1	DIR_NTRes	Reserved. Set to 00h.
000Dh	1	DIR_CrtTimeTenth	Millisecond stamp at file creation
000Eh	2	DIR_CrtTime	Time file was created.
0010h	2	DIR_CrtDate	Date file was created.
0012h	2	DIR_LstAccDate	Last access date
0014h	2	DIR_FstClusHi	High word of the first cluster number of this entry
0016h	2	DIR_WrtTime	Time of last write
0018h	2	DIR_WrtDate	Date of last write
001Ah	2	DIR_FstClusLo	Low word of the first cluster number of this entry
001Ch	4	DIR_FileSize	File size in bytes

The ROM code checks each entry in the root directory until either the booting file is found or the entry is empty (first byte is 00h), or when the end of the root directory is reached. Entries with the ATTR_LONG_NAME attribute (LFN) and first byte at E5h (erased file) are ignored. When found, the first cluster offset of the file is read from the DIR_FstClusHi/DIR_FstClusLo fields. There is a slight difference between FAT12/16 and FAT32 when handling the root directory. On FAT12/16, this directory has a fixed location (see Table 33-47) and length fixed by BPB_RootEntCnt, which is the total of 32-byte entries. Therefore, handling this directory is straightforward. On FAT32, the root directory is like a standard file. The FAT must be used to retrieve each sector of the directory. Step 4 describes the way in which the FAT is handled.

4. Buffer FAT entries in the FAT buffer.

   When the booting file is found, the ROM code reads the FAT and buffers the singly-linked chain of clusters in the FAT buffer that is used during boot to access the booting file directly, sector by sector. For FAT12/16 and for FAT32, multiple copies of the FAT exist (ROM code supports only two copies), after the boot sector.

$$FATn_{sector} = BPB\_HiddSec + BPB\_RsvdSecCnt + BP\_FatSz \cdot n$$

init-E002

   The size of the FAT buffer is given by BPB_FATSz16 or BPB_FATSz32. The ROM code checks each copy of the FAT if the values are identical. If the values are different, the ROM code uses the value from the last FAT copy. With the FAT32 file system, the copy system can be disabled according to a flag in BPB_ExtFlags[7]. If this flag is set, the FAT BPB_ExtFlags[3:0] bit field is used. In this case, no verification is made by the ROM code with other copies of FAT.

   The FAT is a simple array of values, each referring to a cluster in the data area. One entry of the array is 12, 16, or 32 bits, depending on the file system in use. The value in an entry defines whether the cluster is being used or not, and if another cluster must be considered. This creates a singly-linked chain of clusters defining the file. Table 33-48 describes the meaning of an entry.

NOTE: For compatibility, cluster 0 and cluster 1 are not used for files, and these entries must contain:

- FF8h and FFFh (for FAT12)
- FFF8h and FFFFh (for FAT16)
- 0FFFFFF8h and 0FFFFFFFh (for FAT32)

Table 33-48. FAT Entry Description

FAT12	FAT16	FAT32	Description
000h	0000h	00000000h	Free cluster
001h	0001h	00000001h	Reserved cluster
002h–FEFh	0002h–FFEFh	00000002h–0FFFFFEFh	Used cluster; value points to next cluster
FF0h–FF6h	FFF0h–FFF6h	0FFFFFF0h–0FFFFFF6h	Reserved values
FF7h	FFF7h	0FFFFFF7h	Bad cluster
FF8h–FFFh	FFF8h–FFFFh	0FFFFFF8h–0FFFFFFFh	Last cluster in file

NOTE: FAT32 uses only bits [27:0]; the upper 4 bits are usually 0 and must remain untouched.

When accessing the root directory for FAT32, the ROM code starts from the root directory cluster entry and follows the linked chain to retrieve the clusters.

When the booting file has been found, the ROM code buffers each FAT entry corresponding to the file in a sector way. This means each cluster is translated to one or several sectors, depending on how many sectors are in a cluster (BPB_SecPerClus). This buffer is used later by the booting procedure to access the file.

33.3.7.7 SATA Device Boot Operation

The SATA is a host interface dedicated for mass storage memory devices. SATA aims to connect SATA disk drive compliant SSD (SATA solid state drive) or HDD (hard disk drive) for mass storage use case only.

The mass storage device (SSD or HDD) is intended to be connected through SATA standard connector (in case of HDD) or directly soldered on the board (in case of SSD). In the latter case, the SATA mass-storage device is actually permanently attached to the device.

The device embedded SATA host controller has a single port (P0) implementation, and one internal electrical transceiver (SATA_PHY). For more information on the device-embedded SATA (SATA AHCI core and wrapper), see Section 24.8, *SATA Controller*. For more details on SATA subsystem integrated SATA PHY transceiver components, see Section 26.1, *SATA PHY Subsystem*. A system integration block diagram is provided in following section.

33.3.7.7.1 SATA Booting Overview

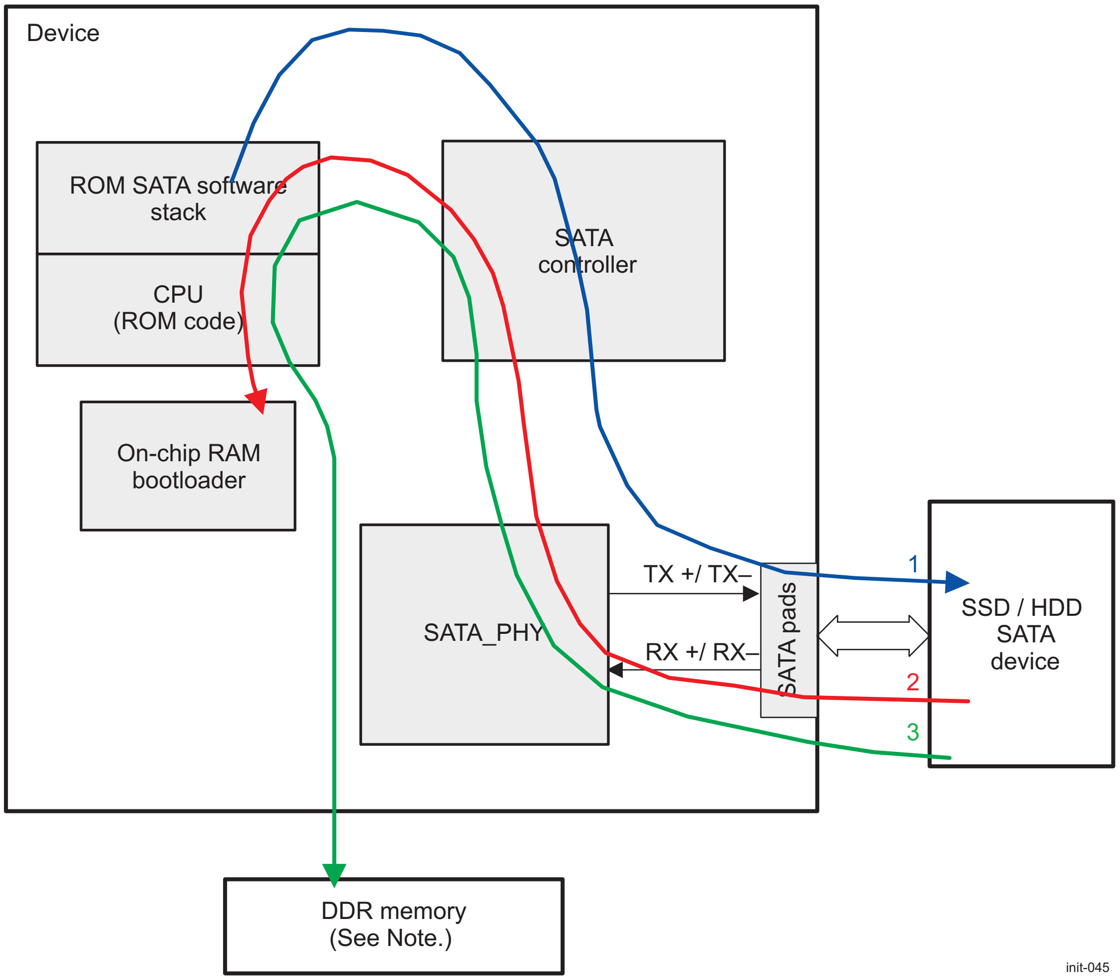
The device attached to the device SATA for boot operation must meet the following requirements:

- It must be compliant with the *Serial ATA Standard* specification, rev. 2.6.
- It must support READ SECTOR(s) ATA command (code 0x20) as defined in the ATA/ATAPI - 7 specification.
- It must be connected directly to the device SATA IOs – sata_tx/sata_ty, sata_rx/sata_ry (no port multiplier connected).

- It must have been already powered by the companion chip or external supply before start of the boot procedure.
- It must be taken in consideration for the device that maximum power on ready time allowed by ROM Code is 900ms. First COMRESET procedure targeting the attached device is triggered 300ms after device SATA subsystem initialization by ROM Code.

Figure 33-31 highlights the booting of the platform from the SSD device.

Figure 33-31. Booting from a Permanently-Attached SSD Device



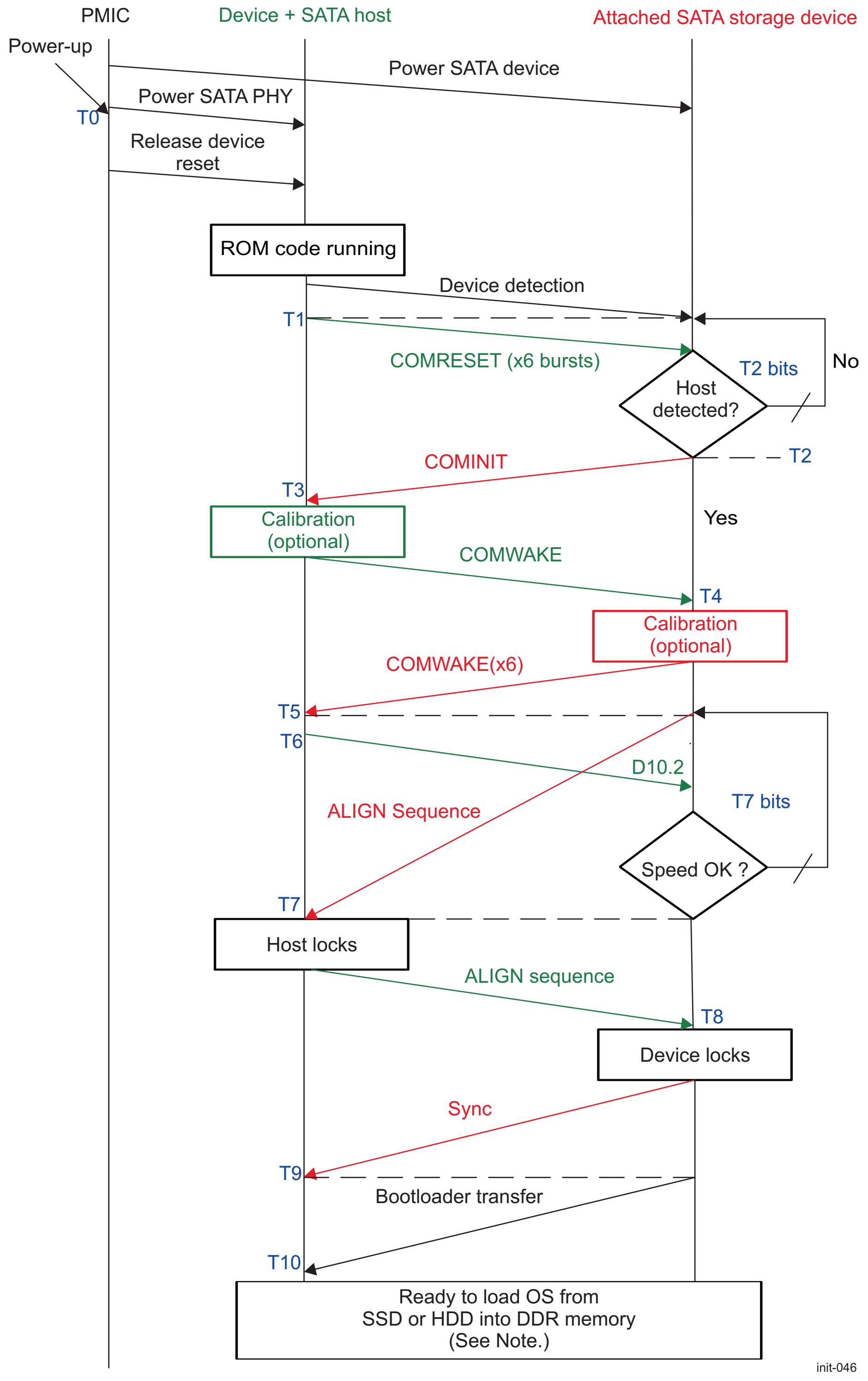
The device booting from an attached SATA mass storage memory device is performed in the following steps:

1. At reset or power on, the device ROM code boots and checks for the SSD/HDD to be ready.
2. The SSD/HDD transfers the boot-loader to the internal RAM of device.
3. The OS in the SSD or HDD is downloaded into DDR memory and the platform is ready.

33.3.7.7.2 SATA Power-Up Initialization Sequence

Figure 33-32 shows all messages exchanges between power-on event up to when the platform is ready. The sequencing aims to depict the complete sequence without any specific connecting issues that may appear.

Figure 33-32. SATA Power-on Initialization Sequencing



The following time phases can be identified during booting from SATA interface:

- T0: the SATA_PHY is powered and device reset is maintained until all Power-management IC companion powers are set.
- T1: the ROM code validates a SATA SSD device detection.
- Within the T1-T2 time interval: the SATA host issues a COMRESET sequence for a minimum of 6 bursts (and a multiple of 6) to force a hardware reset to the SATA peripheral device.

- T2: As long as the attached SATA SSD device does not explicit reset request (COMINIT), the host issues COMRESET sequences
- Within the T2-T3 time interval, once the host release the COMRESET sequence (made of 6 bursts minimum), the device responds with a COMINIT to request a communication initialization (It requests a reset from the host).
- Within the T3-T4 time interval, the SATA host controller may calibrate at T3, but issues a COMWAKE sequence to the peripheral in order to inform the other part the wish to use the link.
- Within the T4-T5 time interval, the SSD device responds and may calibrate. The device response is made of 6 burst COMWAKE sequence.
- Within the T5-T6 time interval, SATA host controller shall start transmitting D10.2 characters no later than a defined moment.
- Within the T6-T7 time interval, when the device host detects the COMWAKE from the device, it starts transmitting D10.2 character at its lowest speed.
- At the same time within T5-T7 time interval, continuous stream of device ALIGN sequence (following the 6 bursts COMWAKE sequence) starting at the device highest speed.
- T7: Without any SATA host’s answer (D10.2 character), the device ALIGN sequence is repeated for as many slower speeds as are supported. When host receives ALIGN sequence, it locks.
- in case no proposed speeds are supported by the host, the device enters in error state.
- in case no ALIGNp sequence is received by the host within a defined time gap after detecting the release of the device COMWAKE, the host restarts the power on sequence indefinitely until stop by user intervention.
- Within the T7-T8 time interval, as soon as the device SATA host locks, it issues an ALIGN sequence to the attached SATA SSD device that also locks
- Within the T8-T9 time interval, the SATA peripheral device sends a SYNC primitive to inform the communication link is established.
- Within the T9-T10 time interval, once the link has been set successfully, the boot loader is transferred from the SATA peripheral device to device internal on-chip RAM

After passing through all above described phases, the device is ready to load OS source from SSD to the device DDR RAM.

33.3.7.7.3 System Conditions and Limitations for SATA Boot

The following system conditions and limitations are defined by ROM Code implementation:

- The ROM Code sets Gen2 speed (3 Gbps) for a SATA boot operation.
- The ROM Code expects that the SATA I/Os are connected to device SATA subsystem at system reset (i.e. a permanently attached boot device).
- The ROM Code expects that the SATA device is powered externally and supplies are set and stable upon entering the booting procedure. The ROM Code does not perform any software action to the companion device with respect to powering the SATA device.
- The ROM Code allows the SATA device 300ms power on ready time before starting the device detection. In case that the detection fails, the ROM Code retries 3 times with 200ms delay before returning and reporting a SATA boot failure.
- The ROM Code does not handle or enable advanced SATA power modes (slumber and partial).

The ROM Code initializes SATA subsystem after SATA boot is started, as defined in this chapter. The ROM Code does not close SATA connection after it was initialized following procedure defined in Section 33.3.4.5, *Booting Device List Setup*.

33.3.7.7.4 SATA Read Sector Procedure in FAT Mode

The booting medium may hold a FAT file system which the ROM Code is able to read and process. The image used by the booting procedure is taken from a specific booting file named "HLO". This file has to be located in the root directory on an active primary partition of FAT12/16 or FAT32 type.

A SATA drive can be configured either as floppy-like or hard-drive-like.

- When acting as floppy-like, the content of the card is a single file system without any **Master Boot Record (MBR)** holding a partition table.
- When acting as hard-drive-like, a **MBR** is present in the first sector of the card. This **MBR** holds a table of partitions, one of which must be **FAT12/16/32, primary and active**.

Refer to the Section 33.3.7.6.5.1, *MBR and FAT File System*, for a more detailed description of FAT file system support.

33.3.8 Image Format

33.3.8.1 Overview

An image has two major parts:

- An optional CH
- Software to execute

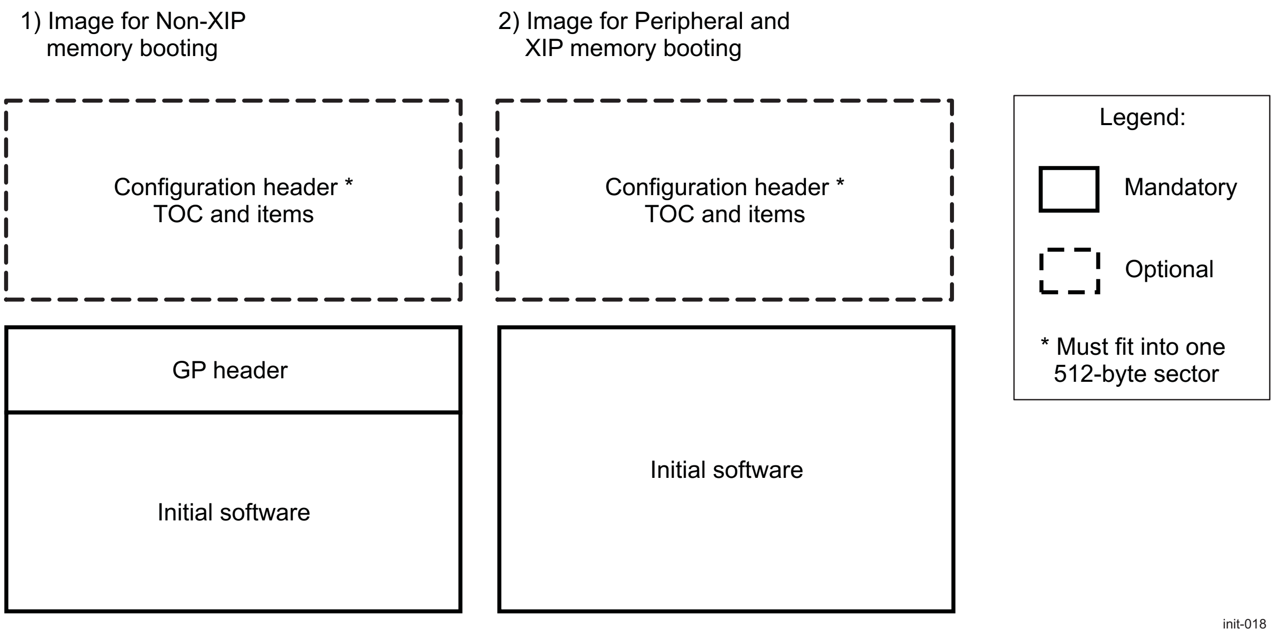
The CH can contain several parameters set by users to speed up booting. It is further described in Section 33.3.8.2, *Configuration Header*.

The second part contains the software that is loaded into the memory and executed.

Figure 33-33 is an overview of the boot image formats. There are two image types:

1. Non-XIP memory booting: This image type is used for memories that require shadowing (for example, NAND). An image for non-XIP memory may not contain a CH and start straight from the GP header. Next, there must be a small header (referred to as a GP header) that contains information about the size and the destination address.
2. When the memory device is of XIP type (for example, NOR), the GP header is not required, and the image can contain code for direct execution. Optionally, the first sector can contain a CH. The same image format is used for peripheral booting (where the code is transferred to internal RAM).

Figure 33-33. Image Formats


1) Image for Non-XIP memory booting
2) Image for Peripheral and XIP memory booting
Configuration header * TOC and items
GP header
Initial software
Configuration header * TOC and items
Initial software
Legend:
Mandatory
Optional
* Must fit into one 512-byte sector
init-018

33.3.8.2 Configuration Header

The ROM code default settings (such as clock frequencies, EMIF, GPMC, MMCHS, or QSPI interfaces) can be tuned by the user by using the CH.

The CH can contain the following parts:

- Settings: Various clock settings (mandatory)
- FLASH: Flash interface (GPMC) settings
- eMMC/SD: MMC2/MMC1 interface settings
- QSPI: QSPI interface settings

The beginning of the CH is a table of contents (TOC), which points to each item. This is described in Figure 33-34. Each TOC item is a simple structure described in Table 33-49. The complete CH (CH TOC and items) should fit in a 512-byte sector.

The ROM code identifies the presence of a CH by reading the first TOC item if it contains a known string (CHSETTINGS, CHFLASH, etc.). Next, the TOC is identified and searched until a 0xFFFF FFFF offset is found. The CH is read and parameters are executed sequentially.

For the sake of simplicity, each field represents the content of a register to be modified. Only fields required for the configuration are used; fields for status, for instance, are not modified and therefore are not shown in the tables.

Figure 33-34. CH Format

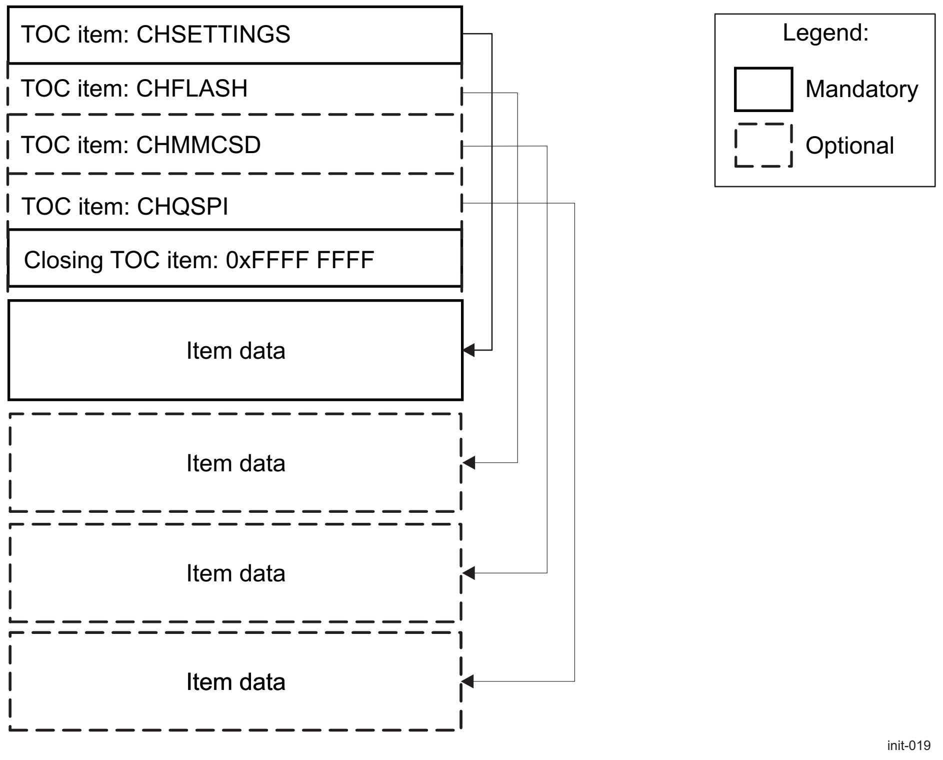


Table 33-49. CH TOC Item

Offset	Field	Size (Bytes)	Description
0x0000	Start	4	Offset from the start address of the TOC to the actual address of item contents
0x0004	Size	4	Size of item
0x0008	Reserved	4	Unused
0x000C	Reserved	4	Unused
0x0010	Reserved	4	Unused
0x0014	Filename	12	12-character name of a item, including the zero termination character(\0)

The ROM Code recognizes sections pointed to by the TOC based on the filename as described in Table 33-50

- The ‘X-LOADER’, ‘2ND’, ‘MLO’, ‘ULO’, or ‘HLO’ section contains the Initial Software
- Optionally, TOC may contain CH sections. The “CHSETTINGS” is a mandatory section of CH and is used to recognize CH presence.

Table 33-50. TOC Filenames

Filename	Item Type	Usage	Description
MLO	Initial Software	eMMC/SD Memory Booting	“Mmc LOader”
HLO	Initial Software	SATA Memory Booting	“Hdd LOader”
ULO	Initial Software	USB or UART Peripheral Booting	“Usb LOader”
2ND	Initial Software	USB or UART Peripheral Booting	“Secondary Loader”
X-LOADER	Initial Software	Other devices	“eXternal LOADER”
CHSETTINGS	Configuration Header	Memory and Peripheral Booting	Configuration Header General Setting Item
CHFLASH	Configuration Header	Memory and Peripheral Booting	Configuration Header GPMC Item
CHMMCSD	Configuration Header	Memory and Peripheral Booting	Configuration Header eMMC/SD Item
CHQSPI	Configuration Header	Memory and Peripheral Booting	Configuration Header QSPI Item

33.3.8.2.1 CHSETTINGS Item

The CHSETTINGS configuration header contains settings specific to the clock system. The ROM code configures the device clocking to some default settings as described in Section 33.3.4.4, *Clocking Configuration*. The CH CHSETTINGS section contains a method to override the ROM code default clock settings.

Table 33-51 describes the fields. The clocking procedure and the clocking setting structure are described in Section 33.3.4.6, *Software Booting Configuration*.

Table 33-51. CHSETTINGS Item

Offset	Field	Description
0000h	Section key	Key used for item verification: C0C0C0C1h
0004h	Valid	Enables/disables the section: 00h: Disable Others: Enable
0005h	Version	Configuration header version 01h Others: Reserved
0006h	Reserved	
0008h	Clocking settings	Described in Table 33-19, *Software Booting Configuration Structure*, starting from the FLAGS field.

33.3.8.2.2 CHFLASH Item

The CHFLASH configuration header contains settings specific to the general-purpose memory controller (GPMC). For more information, see Section 15.4, *General-Purpose Memory Controller*. Table 33-52 describes the fields.

Table 33-52. CHFLASH Item

Offset	Field	Description
0000h	Section Key	Key used for section verification: C0C0C0C3h.
0004h	Valid	Enables/disables the section: 00h: Disable Others: Enable
0005h	Reserved	
0008h	GPMC_SYSCONFIG (LSW)	Register values

Table 33-52. CHFLASH Item (continued)

Offset	Field	Description
000Ah	GPMC_IRQENABLE (LSW)	
000Ch	GPMC_TIMEOUT_CONTROL (LSW)	
000Eh	GPMC_CONFIG (LSW)	
0010h	GPMC_CONFIG1_0	
0014h	GPMC_CONFIG2_0	
0018h	GPMC_CONFIG3_0	
001Ch	GPMC_CONFIG4_0	
0020h	GPMC_CONFIG5_0	
0024h	GPMC_CONFIG6_0	
0028h	GPMC_CONFIG7_0	
002Ch	GPMC_PREFETCH_CONFIG1	
0030h	GPMC_PREFETCH_CONFIG2 (LSW)	
0032h	GPMC_PREFETCH_CONTROL (LSW)	
0034h	GPMC_ECC_CONFIG (LSW)	
0036h	GPMC_ECC_CONTROL (LSW)	
0038h	GPMC_ECC_SIZE_CONFIG	
003Ch	Reserved	

33.3.8.2.3 CHMMCSD Item

The CHMMCSD configuration header contains settings specific to the high-speed MMC/SD/SDIO host controller (MMCHS). For more information, see Chapter 25, *eMMC/SD/SDIO*. Table 33-53 describes the fields.

Table 33-53. CHMMCSD Item

Offset	Field	Description
0000h	Section key	Key used for section verification C0C0C0C4h
0004h	Valid	Enables/disables the section: 00h: Disable Other: Enable
0005h	Reserved	
0008h	CLKD	Functional clock divisor MMCHS_SYSCTL[15:6] CLKD = 0x0: FCLK/1 0x1: FCLK/1 0x2: FCLK/2 … 0x3FF: FCLK/1023 0xFFFF FFFF: Do not modify clock divisor
000Ch	MMCHS interface bus width	Configure the MMCHS interface bus width according to the field value : 1: Configured to 1 bit (SDR) 2: Configured to 4 bits (SDR) 4: Configured to 8 bits (SDR)[1] 8: Configured to 4 bits (DDR)[1] 16: Configured to 8 bits (DDR)[1] 0xFFFF FFFF: Do not update bus width. Others: Reserved

(1) The 8 bit SDR and 4-bit/8-bit DDR (values 4, 8, and 16) are only applicable to eMMC devices.

The ROM code provides a booting parameter structure to the initial software (see Section 33.3.8.4, *Image Execution*). This structure contains a field that indicates whether the configuration header items have been correctly processed. For a CHMMCSD item, if the MMCHS_SYSCTL and bus width fields are set to 0xFFFF FFFF, the booting parameters report that the CHMMCSD section has not been executed, regardless of the value of the Valid field.

33.3.8.2.4 CHQSPI Item

The CHQSPI configuration header contains settings specific to the QSPI interface controller. For more information, see Section 24.5, *Quad Serial Peripherial Interface*. Table 33-54 describes the fields.

Table 33-54. CHQSPI Item

Offset	Register Field Modified	Description
0000h	Section key	Key used for section verification C0C0C0C6h
0004h	Valid	Enables/disables the section: 00h: Disable Other: Enable
0005h	Reserved	
0008h	SPI Clock	SPI clock rate: 0x03: 32 MHz 0x07: 16 MHz 0x13: 48 MHz 0x17: 24 MHz 0x1F: 12 MHz All other: reserved
0009h	RCMD	Read command
000Ah	READ_TYPE	Determines if the read command is a single, dual or quad read mode command
000Bh	NUM_A_BYTES	Number of address bytes to be sent
000Ch	NUM_D_BYTES	Number of dummy bytes to be used for fast read

33.3.8.3 GP Header

When the booting memory device is non-XIP (for example, NAND) the image must contain a small header, located before the executable code, and having the size of the software to load and the destination address of where to store it. Table 33-55 describes the image format. The GP header is not required when booting from an XIP memory device (for example, NOR) or in case of peripheral booting. In this case, the peripheral or memory booting image starts directly with executable code.

Table 33-55. GP Header Image Format

Field	Non-XIP Device Offset	XIP Device Offset	Size (Bytes)	Description
Size	0x0000	–	4	Size of the image (including GP header)
Destination	0x0004	–	4	Address where to store the code or code entry point
Image Code	0x0008	0x0000	x	Executable code

NOTE: The Destination address field stands for:

- Target address for the image copy from the non-XIP storage to the target XIP location (for example, internal RAM or SDRAM)
- Entry point for image code

Users must take care to locate the code entry point to the target address for image copy.

33.3.8.4 Image Execution

The image is executed when the ROM code performs the branch to the first executable instruction in the initial software. In non-XIP, the execution address is the first word after the GP header. The branch is performed in public ARM supervisor mode. The R0 register points to the booting parameter structure that contains information about booting execution. Table 33-56 shows the booting parameter structure.

Table 33-56. Booting Parameter Structure

Offset	Field	Size (Bytes)	Description
0x00	Booting message	4	Last received booting message
0x04	Memory booting device descriptor	4	Pointer to the memory device descriptor that has been used during the memory booting process
0x08	Current booting device	1	Code of device used for booting: 0x01: XIP 0x02: XIP (with wait monitoring) 0x03: NAND 0x05: SD cards 0x06: eMMC (boot partition) 0x07: eMMC 0x09: SATA 0x0A: QSPI_1 0x0B: QSPI_4 0x43: UART 0x45: USB Others: Reserved
0x09	Reset reason	1	Current reset reason bit mask (bit = 1, event present): direct copy from lower byte of PRM_RSTST (more bits exist in PRM_RSTST): [0]: Power-on (cold) reset [1]: Global software warm reset [2]: Reserved [3]: MPU watchdog reset [4]: Reserved [5]: External warm reset [6]: VDD_MPU voltage manager reset [7]: VDD_MM voltage manager reset
0x0A	CH flags	1	Configuration header items flag. Each item is described by 1 bit. A set bit indicates that the item was executed: [0]: CHSETTINGS [2]: CHFLASH [3]: CHMMCSD [4]: CHQSPI Other bits: Reserved

33.3.9 Tracing

Tracing in the public ROM code consists in 32-bit vectors for which each bit corresponds to a particular way point in the ROM code execution sequence. Table 33-57 through Table 33-60 list the organization of the tracing data in RAM. Tracing vectors are initialized at the beginning of the start-up phase and are updated all along the boot process.

There are two sets of tracing vectors (Table 33-14, *Tracing Data)*. The first set is the current trace information (after a cold or warm reset). The second set holds a copy of trace vectors collected at the first ROM code run after a cold reset. As a result, after a warm reset it is possible to have visibility on the boot scenario that occurred during a cold reset.

Table 33-57 lists the organization of tracing vector 1.

Table 33-57. Tracing Vector 1

Bit	Group	Meaning
0	Boot	Passed the public reset vector
1	Boot	Entered main function
2	Boot	Running after the cold reset
3	Boot	Main booting routine entered
4	Memory boot	Memory booting started
5	Peripheral boot	Peripheral booting started
6	Boot	Booting loop reached last device
7	Boot	GP header found
8	Peripheral Boot	Booting message Skip peripheral booting received
9	Peripheral Boot	Booting message Change device received
10	Peripheral boot	Booting message Peripheral booting received
11	Peripheral boot	Booting message Get ASIC ID received
12	Peripheral boot	Device initialized
13	Peripheral boot	ASIC ID sent
14	Peripheral boot	Image received
15	Peripheral boot	Peripheral booting failed
16	Peripheral boot	Booting message not received (time-out)
17	Peripheral boot	Image size not received (time-out)
18	Peripheral boot	Image not received (time-out)
19	Reserved	
20	Boot	Configuration header found
21	Boot	CHSETTINGS item processed
22	Boot	Reserved
23	Boot	CHFLASH item processed
24	Boot	CHMMCSD item processed (clock)
25	Boot	CHMMCSD item processed (bus width)
26	Boot	CHMMCSD item processed (eMMC DDR mode)
27	Reserved	
28	Boot	SWCFG general detected
29	Boot	SWCFG clocks detected
30	Boot	SWCFG time-out detected
31	Reserved	

Table 33-58 lists the organization of tracing vector 2.

Table 33-58. Tracing Vector 2

Bit	Group	Meaning
0	Configuration Header	CHSATA item processed
1:3	Reserved	
4	USB	USB connected
5	USB	USB configured
6:7	Reserved	
8	Configuration Header	CHQSPI item processed (clock speed)
9	Configuration Header	CHQSPI item processed (Read command)
10:11	Reserved	Reserved
12	Memory boot	Memory booting trial (first block)
13	Memory boot	Memory booting trial (second block)
14	Memory boot	Memory booting trial (third block)
15	Memory boot	Memory booting trial (fourth block)
16:29	Reserved	
30	Boot	Jumping to Initial Software
31	Reserved	

Table 33-59 lists the organization of tracing vector 3.

Table 33-59. Tracing Vector 3

Bit	Group	Meaning
0	Reserved	
1	Memory boot	Memory booting device XIP
2	Memory boot	Memory booting device XIPWAIT
3	Memory boot	Memory booting device NAND
4	Reserved	
5	Memory boot	Memory booting device SD
6	Memory boot	Memory booting device: eMMC (from boot partition)
7	Memory boot	Memory booting device MMC2: eMMC (from user area)
8	Reserved	
9	Memory boot	Memory booting device: SATA
10	Memory boot	Memory booting device: QSPI_1
11	Memory boot	Memory booting device: QSPI_4
12:18	Reserved	
19	Peripheral boot	Peripheral booting device UART3
20	Reserved	
21	Peripheral boot	Peripheral booting device USB
22:27	Reserved	
28	Boot	Reserved
29	Boot	Reserved
30:31	Reserved	

Table 33-60 lists the organization of tracing vector 4.

Table 33-60. Tracing Vector 4

Bit	Group	Meaning
0:23	Reserved	
24	MMC/SD	SD card detected PBIAS configuration is 1.8V
24:27	Reserved	
28	SATA	SATA is configured
29	SATA	SATA retried
30	SATA	SATA failed
31	Reserved	

33.4 Services for HLOS Support

The ROM code provides different services that can be called for L1 and L2-cache maintenance, Enter in Low Power, etc. These services are implemented in monitor mode and must be called by using the SMC instruction. The caller must ensure the save and restore of the processor registers before and after calling the Monitor Service.

The following code example shows how the monitor ROM code functions can be accessed by an application running in public mode:

```
;------------------------------------------------------------------------------
; FUNCTION: SetL2CacheLatency
;
; DESCRIPTION: Function calls the Monitor Service to setup the L2 Cache Latency
;
; INPUTS:  r0 Tag RAM Latency to set
;          r1 Data RAM Latency to set
; RETURN:
;
;------------------------------------------------------------------------------
SetL2CacheLatency           FUNCTION

  PUSH {R1-R12, LR}
  LDR R12, =0x105
  SMC 0x1
  POP {R1-R12, PC}
  ENDFUNC
```

33.4.1 Hypervisor

Table 33-61. Start Hypervisor

Function ID	**Description**	
R12 = 0x102	This function starts the CPU Hypervisor mode.	
Parameters		
Type	**Location**	**Description**
Input	R0	start address of the Hypervisor software
Output		
Return		

33.4.2 Caches Maintenance

Table 33-62. Clean L1 and/or L2 cache

Function ID	**Description**	
R12 = 0x103	This function cleans and invalidates the CPU L1-cache and if requested in the input parameter R0, clean the full L2-cache.	
Parameters		
Type	**Location**	**Description**
Input	R0	if not equal to 0, clean L2-cache
Output		
Return		

33.4.3 CP15 Registers

Table 33-63. Write L2 Cache Auxiliary Control

Function ID	Description	
R12 = 0x104	This function writes the CP15 L2 Cache Auxiliary Control Register with the input given value in R0.	
Parameters		
Type	**Location**	**Description**
Input	R0	Value to set in the register
Output		
Return		

Table 33-64. Write Tag and Data RAM Latency Control Register

Function ID	Description	
R12 = 0x105	This function writes the L2 Cache Tag and Data RAM Latency in the CP15 L2 Control Register with the input given value in R0 and R1.	
Parameters		
Type	**Location**	**Description**
Input	R0 R1	Tag RAM Latency to set Data RAM Latency to set
Output		
Return		

Table 33-65. Write L2 Cache Prefetch Control Register

Function ID	Description	
R12 = 0x106	This function writes the CP15 L2 Cache Prefetch Control Register with the input given value in R0.	
Parameters		
Type	**Location**	**Description**
Input	R0	Value to set in the register
Output		
Return		

Table 33-66. Write Auxiliary Control Register

Function ID	Description	
R12 = 0x107	This function writes the CP15 Auxiliary Control Register with the input given value in R0.	
Parameters		
Type	**Location**	**Description**
Input	R0	Value to set in the register
Output		
Return		

33.4.4 Wakeup Generator

Table 33-67. Write AMBA IF Register

Function ID	Description
R12 = 0x108	This function writes the WakeupGen AMBA I/F Register with the input given value in R0.
Parameters	

Table 33-67. Write AMBA IF Register (continued)

Function ID	Description	
Type	**Location**	**Description**
Input	R0	value to set in the register
Output		
Return		

33.4.5 ARM Timer

Table 33-68. Set Timer CNTFRQ Register

Function ID	Description	
R12 = 0x109	This function set the ARM Timer CNTFRQ (CP15 register) with the input given value in R0.	
Parameters		
Type	**Location**	**Description**
Input	R0	value to set in the register
Output		
Return		