Marvell Solutions

Marvell® PXA3xx Processors and Tavor P Processor

Boot ROM Reference Manual

Doc. No. MV-S301208-00, Rev. -
January 29, 2008
Document Classification: PUBLIC RELEASE

MOVING FORWARD FASTER®

MARVELL®

Document Conventions

Note

Provides related information or information of special importance.

Caution

Indicates potential damage to hardware or software, or loss of data.

Warning

Indicates a risk of personal injury.

Document Status

Doc Status: Preliminary	Technical Publication: 2.3x

For more information, visit our website at: www.marvell.com

Table of Contents

List of Tables

1 Boot ROM Functional Overview

1.1 General Description

The Boot ROM software is preloaded into the processors' internal ROM. The Boot ROM is an enabling component of the Marvell Trusted Boot solution. The Boot ROM provides support for the implementation of processors with Intel XScale® technology for the usage cases described in the following sections. When operating in a non-trusted mode, the Boot ROM supports loading software from various devices as part of the boot process. When operating in a trusted mode, the Boot ROM is considered the root of trust for the platform and it handles the initialization of the Marvell® Wireless Trusted Module (Marvell® WTM) subsystem.

Because the Boot ROM is configured at manufacture, no changes can be made to the boot configuration of the Boot ROM. The correct boot configuration must be ordered from Marvell for a given platform architecture however some processors support an "auto-boot feature and are not configured to one boot memory for the first boot. Refer to 2 "Boot ROM Differences" for details on auto-boot support.

The list of supported boot configurations and how to configure the PXA31x auto-boot feature are discussed in this document.The Boot ROM implements a common set of functionality across all implementations of the PXA3xx processor family and Tavor processor. Refer to the specific processor developers manual for details of the processor features.

1.2 Purpose Of This Document

This document covers the operational details of the Boot ROM for the PXA3xx processor family and Tavor processors. System- level dependencies are also covered such as:

- Booting the platform
- Choice of NAND memory and how the Boot ROM supports NAND
- Managed NAND support - This is NAND memory with a NOR-like bus interface such as Samsung OneNAND and Sandisk mDOC.
- Software requirements for both trusted and non-trusted platforms
- Boot ROM versions and feature sets
- Exiting low power modes
- Host Tools
- JTAG Re-enablement

1.3 ROM Location, Size, and Mapping

The processor family has 48 KB of internal ROM, which is used for the Boot ROM. Anytime the processor goes through a power transition that causes a jump to the reset vector, the Boot ROM is mapped to two different address spaces, via hardware mechanisms:

- `0x0000_0000 - 0x0000_BFFF`
- `0x5E00_0000 - 0x5E01_FFFF`

The physical ROM is mapped to the `0x5E00_0000 - 0x5E01_FFFF` address space in the processor memory map. The `0x0000_0000 - 0x0000_BFFF` address space is a virtual memory mapping implemented by the hardware to locate the vector table at the correct address for the processor core. After any power transition which causes the processor core to jump to the reset vector, the Boot ROM is the first code to execute on the processor.

While the Boot ROM is running and is mapped to `0x0000_0000 - 0x0000_BFFF`, external flash memory mapped to this region is not accessible. Before handing control to any image, the Boot ROM remaps the `0x0000_0000 - 0x0000_BFFF` address space to external flash memory on chip select 0 to make access to this flash memory possible. Once this address space is remapped to the external flash, higher layers of software must accommodate the vector table by setting up a vector table in an appropriate location depending on the usage model chosen. Refer to the specific processor developers manual for more details on the requirements for setting up the vector table.

1.4 Boot ROM Overview

After reset, the Boot ROM performs the essential initialization including programming the clocks, GPIO settings, and the interrupt controller. The Boot ROM verifies whether the reset reason was a return-from-hardware reset (HWR), Watchdog reset (WDR), or a resume-from-S3 power-state reset.

Note

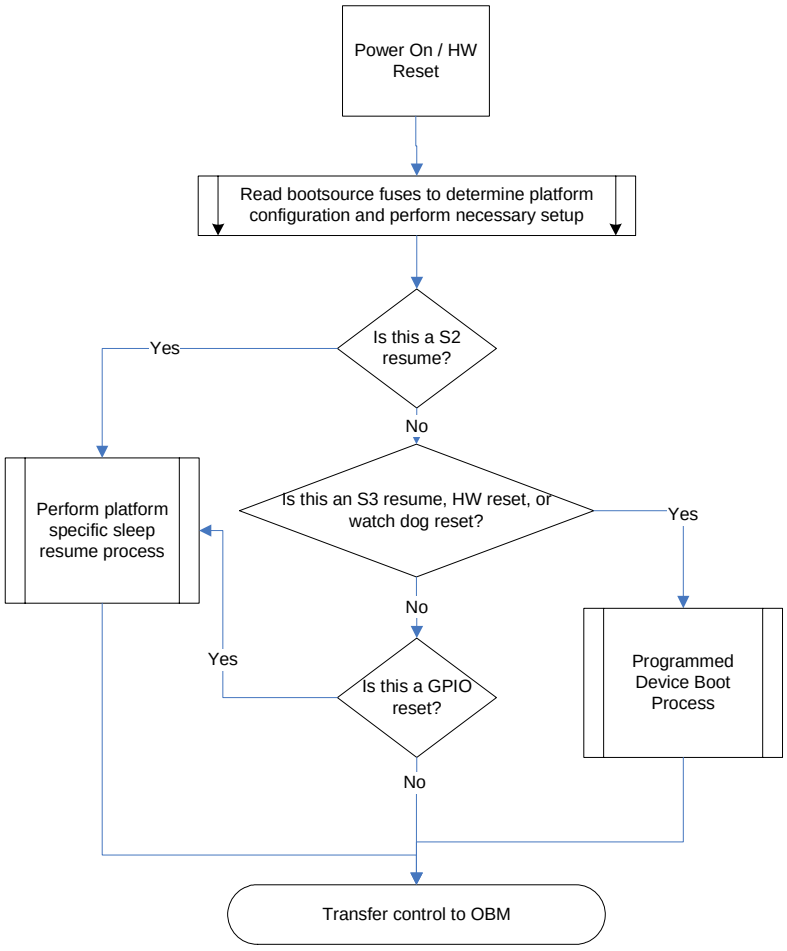
After V3.20 Boot ROM, the Boot ROM does not perform verification to identify the last reset transition. All Version 2.xx Boot ROMs perform this task. Therefore, for the application processors, only the PXA320 and PXA30x A1 perform this procedure as indicated in Figure 1.

If the reset is not attributable to any of these reasons listed above, the Boot ROM uses the platform configuration data that is provided by the bootsource fuses to determine how to resume the platform from an S2/D3/C4 state or GPIO reset.

For more details, refer to Chapter 2 and Chapter 3 of this document.

Figure 1 shows the execution flow of the Boot ROM from reset. All processor resets are directed to the Boot ROM, from where the appropriate path to resume or boot is determined.

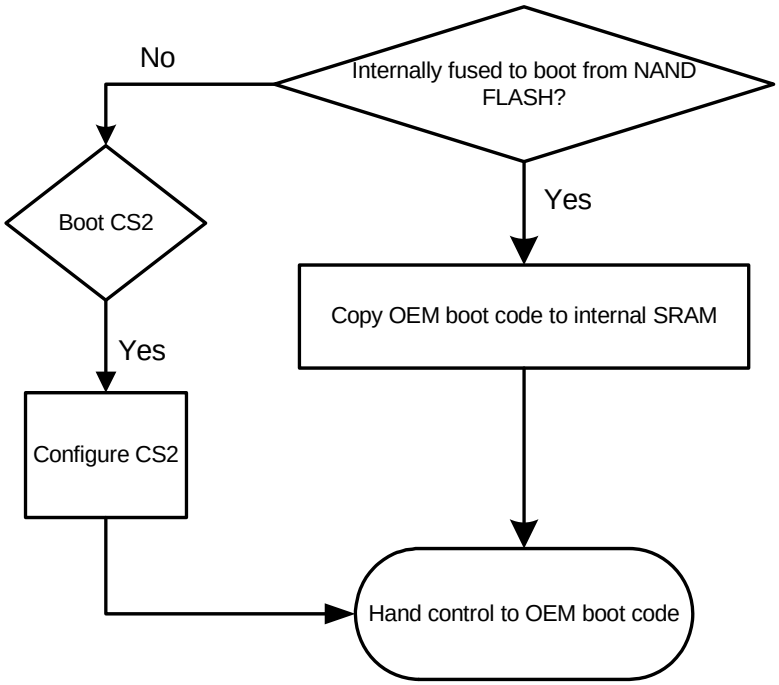
Figure 1: PXA3xx Top Level Boot ROM Flow



1.4.1 Boot Process for Programmed Device

The Boot ROM identifies the platform configuration fuses and transitions to the external flash. For platforms using NAND flash, the OEM boot module (OBM) is copied from Block 0 of the NAND flash to internal SRAM at and control is transferred to the OBM. For platforms using NOR flash, control is transferred to the external flash on nCS2 of the DFI bus. See Figure 2 (Boot Process Flow Chart).

Figure 2: Boot Process Flow Chart



There are many more details that are noted in later chapters. This section is the introduction and there are differences between the Boot ROM revisions and the different processor silicon steppings; however the main flow of execution is the same.

2 Boot ROM Differences

This section provides information on the differences between Boot ROM version 2.x and 3.x. The Boot ROM is implemented in all of the PXA3xx family processors and the Tavor processor. Table 1 describes which processor is implemented with the different versions of the Boot ROM.

Table 1: Version 2.xx and Version 3.xx High Level Differences

Processor Stepping	**PXA32x B1/B2; Tavor A0**	**PXA32x C0**	**PXA301 A1 and PXA300 A1**	**PXA31x A1; Tavor B0.a**	**PXA31x A2; PXA302; PXA312 PoP Tavor B0.b**
Boot ROM Version	**Version 2.22**	**Version 3.38**	**Version 3.19**	**Version 3.27**	**All Version 3.32 except PXA31x A2 Version 3.33**
	Non-Trusted				**Trusted and Non-Trusted**
JTAG Re-enablement (only for trusted platforms)[1]	No	No	No	No	Yes
Download via USB1.1 (differential/SE)	No	No	No	Not applicable for PXA31x A1; Yes (default) for Tavor B0.a	Yes (default) except for PXA31x A2
Download via USB2.0	No	No	No	Yes (default) for PXA31x; Yes with (TIM)/ (NTIM) override but only for Tavor B0	Not applicable for PXA302; Yes with TIM/NTIM override but only for Tavor B0; Yes (default) for PXA310 A2
Download via USB OTG	No	No	No	No	No
Download via UART	No	No	No	Yes	Yes
TIM/NTIM Support	No	No	NTIM	NTIM	TIM/NTIM
					1. The **BootFlashSign** field as defined in the TIM/NTIM header must be included. Previous Boot ROM versions did not use this field. **2. ImageSizeToCRC** must be 0x00
			V3.xx Boot ROM can also operate without a TIM/NTIM header. Using the V2.xx IVM/OBM method will also work.		

Table 1: Version 2.xx and Version 3.xx High Level Differences (Continued)

Processor Stepping	PXA32x B1/B2; Tavor A0	PXA32x C0	PXA301 A1 and PXA300 A1	PXA31x A1; Tavor B0.a	PXA31x A2; PXA302; PXA312 PoP Tavor B0.b
Boot ROM Version	Version 2.22	Version 3.38	Version 3.19	Version 3.27	All Version 3.32 except PXA31x A2 Version 3.33
	Non-Trusted				Trusted and Non-Trusted
Tamper Recovery	No	No	No	No	Yes done in WTPSP
Sandisk M-DOC H3 and Samsung OneNAND	Yes, using XIP area	Yes, using XIP area	Yes, using XIP area	Yes, using XIP area	Yes, with NTIM and TIM support
	The drivers are not integrated into the Boot ROM. The processor initially sees these devices as NOR memory. The OBM boot module can be loaded from XIP area, then the driver can be loaded after.				The drivers are integrated into the Boot ROM. At Boot time the processor can fully access the managed NAND memory. Refer to Section 5.5 and Section 5.6 for details.
NAND Interface Timing Configuration - Programmed in NDTR1CS0[tR]	39.23uS	39.23uS	38.76uS	Tavor - 39.23uS PXA31x A1 - 38.76uS	52.4uS
MMC/SD Boot	No	No	No	No	No
Auto-Boot	No	Yes[2]	No	Yes	Yes
Boot from nCS0[3] (Static CS)	No	No	No	PXA31x only	PXA31x only
ONFI[4] Compliant NAND Reset Command 0xFF	No	Yes	No	No	Yes

1. The JTAG port is enabled on all non trusted platforms. Refer to JTAG re-enablement in Section 9.3, JTAG Re-enablement, on page 100.
2. Version 3.xx Auto-Boot mode is not backward compatible with Version 2.xx. See Note in Auto-Boot in Non-Trusted Platforms.
3. XIP NOR only, not managed NAND (OneNand + mDOC)
4. ONFI is the Open NAND Flash Interface specification www.onfi.org

2.1 Non-Trusted vs Trusted

Version 2.xx implements only the non-trusted boot process. A non-trusted platform is defined as a platform that does not use the Wireless Trusted Module (WTM) to validate the integrity of the OS image.

Version 3.xx supports both trusted and non-trusted boot process. A trusted platform does use the WTM for security encryption and validation of an uncompromised OS image.

2.2 TIM/NTIM/OBM Support

2.2.1 Version 2.xx Platforms

Version 2.xx platforms have-hard coded requirements for booting to the OEM boot module (OBM) image. These requirements impose several restrictions on the platform for NAND boot, specifically:

- A maximum size limit of two blocks can be used for OBM, restricting the size
- Two words of offset must be pre-pended to the OBM before programming to the NAND Flash
- For an OBM of one block in size:

1. Set NAND Flash address 0x0 to 0x0000_0008.
2. Set NAND Flash address 0x4 to 0x0000_0000.
3. Burn the OEM boot module image to NAND Flash address at 0x8.

- For an OBM of two blocks in size, follow the same as the steps above for OBM of one block in size; however, set NAND Flash address 0x4 to address of Block 2.
- All unused pages within Block 0, after the OBM image, must be programmed with 0x0.
- Runtime location is fixed in the ISRAM
- Use of an NTIM is not supported

Note

If the first two words are not written as 0x0 and the unused pages in Block 0 are also not written as 0x0, then the system fails to boot. Not performing this action creates inconsistent Error-Correcting Code (ECC) information so the Boot ROM aborts the boot process.

2.2.2 Version 3.xx Platforms

Version 3.xx trusted boot platforms use the TIM and NTIM. These headers store information about the flash layout and runtime locations for the platform software. The header format is compatible between trusted and non-trusted systems allowing reuse of images. The difference between a TIM and an NTIM is the type of information within the header. The TIM holds security information in addition to the flash layout and runtime information.

- When a header (NTIM or TIM) is used, the requirement to program all unused pages as specified for V2 Boot ROM is no longer necessary. The header contains the size of the OBM image. If not using a header, Marvell recommends that the unused pages be programmed with 0x0.

V3.xx Boot ROM also has a backward compatibility mode such that the OBM/IVM from V2 may be used as an option.

2.3 Boot ROM Address Maps

2.3.1 ISRAM Usage

This section shows the internal SRAM address usage. Approximately 4 KB of internal SRAM was freed up in the data/stack area of the Boot ROM and made available for OBM downloading in V3.xx.

Table 2: Version 2.xx ISRAM Usage

ISRAM Address	Size	Usage
0x5C00_0000	32 KB	Caddo

Table 2: Version 2.xx ISRAM Usage (Continued)

ISRAM Address	Size	Usage
0x5C00_8000	48 KB	Boot ROM data/stack
0x5C01_4000	Rest of ISRAM	Image download

Table 3: Version 3.xx ISRAM Usage

ISRAM Address	Size	Usage
0x5C00_0000	32 KB	Caddo
0x5C00_8000	44 KB	Boot ROM data/stack
0x5C01_3000	Rest of ISRAM	Image download

2.3.2 OBM Usage

Version 2.xx has fixed offsets for OBM images based on flash type used. The area in the Static Chip Select 0 (nCS0) from 0x0 to 0xBFFF is dual-mapped to either Boot ROM or external flash, which can make programming difficult. The Boot ROM also uses the first two words for its own operations so the OEM system boot module (OSBM) images had to be manually shifted when programming.

Table 4: Version 2.xx OBM Usage

Boot Device	OBM location in device	Runtime location of OBM
Chip Select 2 - nCS2	Block 0 offset 0x0	0x1000_0000
NAND device on ND_nCS0	Block 0 offset 0x8	0x5C01_4000

Version 3.xx is backward compatible with version 2.xx. Certain requirements, such as the boot state fuses, must be programmed and the offsets in Table 5 must be adhered to. For NAND Flash, the OBM size is limited to one block.

Table 5: Version 3.xx OBM Usage

Boot Device	OBM location in device	Runtime location of OBM
Chip Select 0 - nCS0	block 0 offset 0xC000	0x0000_C000
Chip Select 2 - nCS2	block 0 offset 0x0	0x1000_0000
NAND device on ND_nCS0	block 0 offset 0x0	0x5C01_3000

2.4 PXA31x and Tavor Processor Auto-Boot

All PXA3xx processors and Tavor processors have been pre-programmed prior to customer shipments, which requires customers to specify the required boot memory such as XIP NOR Flash boot from nCS2 and x8 or x16 NAND boot devices.

However, the PXA31x A1 and Tavor B0 stepping do not operate in this way. These processors are shipped without the boot type fuses configured, which allows flexibility in customer builds for using the same device in multiple platforms with different boot-memory configurations.

Auto-boot is a process whereby the Boot ROM probes all the valid boot devices for a valid header file (either a TIM or NTIM). Once this header is found, it is used to boot the system. The Boot ROM does not search for multiple headers. The first valid header that is found is used to boot the system.

The boot memories are probed in the following order:

- XIP on nCS2
- XIP on nCS0 (XIP NOR only; not Managed NAND + mDOC)
- x16 NAND on ND_nCS0
- x8 NAND on ND_nCS0
- Samsung OneNAND on nCS2
- Sandisk mDOC on nCS2

If a valid TIM or NTIM is not found, then the Boot ROM waits for a download operation over USB or UART. This operation would be downloading an image from a host over a USB or UART to the platform. If the TIM or NTIM is not found, this would indicate that the flash memory is not programmed.

Note

If the Boot ROM does not find a valid header and the image download is successful, the subsequent boot probes each boot memory (as described above) but now the Boot ROM finds the valid TIM/NTIM and the boot process continues.

2.4.1 Auto-Boot on Non-Trusted Platforms

The auto-boot process described in Section 2.4, PXA31x and Tavor Processor Auto-Boot occurs for every boot or reset exit on non-trusted platforms. Below is the list of resets and mode exits:

- SOD - start of day
- Hardware reset
- Watchdog reset
- GPIO reset
- S2/D3 exit**
- S3/D4 exit

**Software may optionally configure a D3 resume to internal SRAM. If enabled, then the “probing” is bypassed. This option is enabled in the NTIM/TIM package.

Note

For PXA32x C0 processor, Auto-Boot feature does not allow for backward compatibity between Boot ROM version 2.xx and version 3.xx. In Auto-Boot mode, the PXA32x C0 processor requires the same memory addresses as the PXA31x processor. The PXA32x C0 is fully backward compatible with PXA32x B2 in fused mode only.

2.4.2 Auto-Boot on Trusted Platforms

The auto-boot process described in Section 2.4, PXA31x and Tavor Processor Auto-Boot occurs only for the first boot if a valid TIM is found for a Trusted Boot. If the Boot ROM fails to find a valid header, then the Boot ROM waits for an image download to occur.

Once a TIM is found and has been successfully validated, the platform is bound, which means that the fuses have been configured permanently. All subsequent boots are fused boots and the probing (auto-boot feature) of each memory device does not occur.

The auto-boot feature has certain platform requirements that must be enabled. For fuses to be configured, VCC_MVT must be raised to 1.9V through a sequence of PWR I2C commands sent to the PMIC over the PWR_I2C bus.

A trusted platform must be architected such that the Services Power Management Unit sends I2C commands to the PMIC. The processor sends these command sequences automatically. There are internal handshaking and acknowledgement tokens passed between Caddo, the Boot ROM, and the Services unit to enable the correct sequencing of I2C commands and voltage changes to VCC_MVT. This sequencing is isolated and cannot be reconfigured or changed in any way.

The basic steps of operation are as follows:

1. The Auto-Boot procedure selects the valid boot memory by the Boot ROM finding a valid TIM header.
   - VCC_MVT is currently at default value -> 1.8V
2. MDTV2 is loaded with the value 0x08 -> for 1.9V.
   - The higher voltage is required to allow the fuses to be configured correctly.
   - VCC_MVT operating at 1.9V is only allowed while the fuses are been configured.
3. VCC1[MVS] and VCC1[MGO] are both set to 0b1.
4. Now VCC_MVT voltage is raised by the PMIC to 1.9V.
   - Internal handshaking indicates the fuses have been configured.
   - The Services Unit must now return VCC_MVT to default value.
5. MDTV1 is loaded with 0x04 -> for 1.8V.
6. VCC1[MVS] is cleared 0b0 and VCC1[MGO] is set to 0b1.
7. Now VCC_MVT is lowered to 1.8V.

Refer to the “Clock Controllers and Power Management” chapter of the *PXA3xx Processor Family Developers Manual* for more information on PWR_I2C commands.

Note

Not only must a PXA3xx-compliant PMIC be used in any PXA3xx-enabled platform but also for trusted platforms, the system design must be connected in such a way to enable the PMIC to raise VCC_MVT for fuse configuring. This design may require detailed inspections of the PMIC datasheet to ensure that VCC_MVT is connected to a dynamically programmable PMIC output regulator. This is known as “Boost Mode”. Refer to the *PXA3xx EMTS* for more details.

3 Software Requirements

This chapter lists the requirements that must be performed by the OBM for the NAND and NOR platforms.

3.1 Size Restrictions for the Device Keying Binary

The Device Keying Binary should be restricted in size, based on the available internal SRAM on the processor used. Refer to the internal SRAM usage in Section 2.3.1, “ISRAM Usage”. Marvell suggests that the Device Keying Binary size be restricted to less than 64 KB, based on the minimum internal SRAM that is used (128 KB). Of that 128 KB of internal SRAM, the Marvell® WTM locks 32 KB, which leaves 96 KB for software usage in a minimum internal SRAM scenario.

3.2 OEM Boot Module Requirements for NAND Platforms

The OEM boot module is responsible for performing all of the relevant tasks required for booting the platform. OEMs can choose between using their proprietary mechanism for identifying and loading the images from the NAND flash, or using the Marvell proposed bad block/relocation table located in Block 0 of the NAND flash in identifying and loading images from the flash. An example of an OEM boot module that uses the Marvell proposed bad block/relocation table is provided as a template. The requirements that must be completed by the OEM boot module include the following:

- Identifying the OS loader/image in the NAND flash
- Loading the content of the OS loader/image from the NAND flash into either internal SRAM or the DDR memory, as applicable
- Integrity checking the OS loader/image for trusted platforms
- Executing the loaded image.

The OEM boot loader must first initialize the DDR memory when loading to DDR memory. The OEM boot module must also relocate the contents of a block that goes bad in the process of accessing the block to a new block and update the bad block/relocation table of the platform.

The OEM boot module in Block 0 must also have all the basic OS startup capabilities, such as identifying reset reason and knowing when/how to identify and load the OS loader/image. For example, when returning from sleep resume, the OEM boot module simply performs the relevant requirements or workarounds, and transitions to the OS image in the DDR; when returning from hardware reset, the OEM boot module must perform a full DDR memory initialization and load the OS contents from the NAND flash to the DDR memory before transitioning to the DDR memory to continue execution.

3.2.1 NAND OEM Boot Module Size Restrictions

If the Marvell bad-block management table is used and the Trusted Image Module or image module is used, then the OEM boot module is restricted by the size of the available internal SRAM. If the Marvell bad-block management table or the image module is not present, then the OEM boot module is restricted to Block 0 of the NAND.

3.3 NAND Bad/Relocation Block Table Definition

This section defines the bad-block relocation table that is used by the Boot ROM to load the OEM boot module and by the OEM boot module to identify and load the OS loader/image from the NAND

flash to either the DDR memory or internal SRAM. The bad-block management scheme consists of two components: the bad-block table and the pool of reserved relocatable blocks. The relocation table always resides starting with the last page of Block 0. This offset in Block 0 for the initial table depends on the number of bytes per page and number of pages per block, as shown in Table 6.

Table 6: Relocation Table Addresses

Block 0 Offset Of Relocation Table Base	Block Size	Number of Bytes/Page and Pages/Block	Comments
0x0000_F800	64 KB	2048/32	—
0x0001_F800	128 KB	2048/64	Largest supported block size.

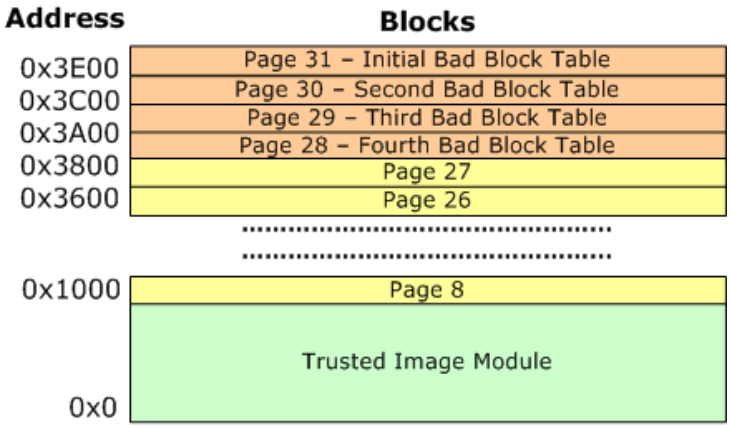
The bad-block table requires exactly one page per block. If the bad-block table has to change at run time, each page is treated like a new slot for additional tables. Rather than erasing and creating a new table over the initial page each time, a new table is simply written to the page directly below the current table, which reduces wear and tear on the block by reducing the number of erase cycles. The maximum number of pages reserved for bad-block tables is 24. When 24 pages have been filled with bad-block tables, the next entity to create a new bad-block table then erases the block, reprograms any non-bad-block table-related information, and creates a new bad-block table at the initial page as defined in the addresses in Table 6.

Note

Note

This implementation requires a binary search algorithm to search for and find the most current table. Figure 3 is an example of a typical Block 0 layout at run time indicating how the slot-based mechanism works.

Figure 3: Block 0 Layout on a Samsung K9K1216Q0C* with 16-KB Block Sizes and 512-Byte Pages



3.3.1 Bad Block Table Definition

Each bad-block table has a layout in flash, as defined with the following structure:

```
Typedef struct S_Reloc
{
    USHORT Header;
    USHORT NumReloc;
    Rel_T  Relo[NAND_RELOC_MAX];
}Reloc_T;
```

The header is a fixed value of `0x524E` to identify the presence of a bad-block table; that is, if the header is valid as defined above, the initial block scan has been completed. Otherwise, the block scan has not been completed. The `NumReloc` parameter identifies the number of blocks that has currently been relocated and is followed by up to 127 relocation pairs.

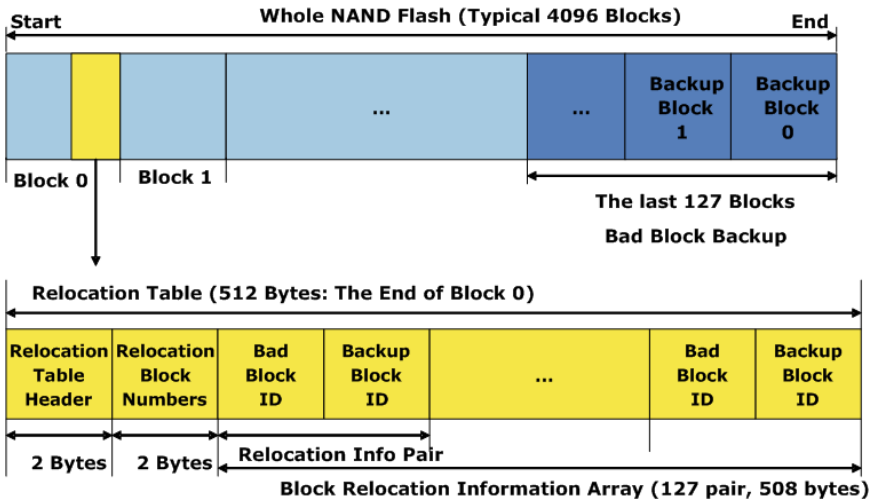
```
Typedef struct S_Rel
{
    USHORT From;
    USHORT To;
}Rel_T;

Const ULONG NAND_RELOC_MAX   = 127;
```

Each “From” entry identifies the block that has been relocated and the entry “To” identifies the relocated block number.

3.3.2 Bad Block Relocation Area

The last 127 blocks of the device are reserved for bad-block relocations. The first block that is relocated goes to the very last block of the device; the second block relocated goes to the second to the last block of the device, and so forth. This process effectively allows relocated blocks to grow from the highest address down. A block in the relocation pool itself may be relocated, so use caution when relocating to skip over these blocks. Figure 4 presents a typical flash part layout and a relocation table layout to tie the concepts together.

Figure 4: Example of Bad Block Table NAND Flash Mapping in Use — Small Block NAND Flash Type: Samsung K9K1216Q0C* (Device ID = 0x46)



3.4 OEM Boot ROM Requirements for NOR Platforms

3.4.1 Traditional Boot Platforms

The OEM boot module on a traditional boot platform is responsible for loading the next layer of software and performing any platform initialization required to boot the operating system and/or communication layer.

3.4.2 Trusted Boot Platforms

The following are the list of tasks that must be performed by the OEM boot module during normal trusted platform boot-up and resume operations.

- Use the OEM proprietary scheme to validate the security of all OEM supplementary binaries (OS/Application/Data, and so on).
- Use the OEM proprietary scheme to validate the security of the mobile operator's service provisioning module, if any.
- For the Tavor processor, the OEM boot module must integrity-check the communications content, initialize the DDR memory, copy the communications content to the DDR memory, and trigger the execution of the communications subsystem.
- Boot the platform.

3.5 Marvell® Wireless Trusted Module Driver Requirements

The Marvell® WTM driver is responsible for completing the following as part of the virgin boot requirements:

- Generate a new Random Number Generator (RNG) seed using the Marvell® WTM from the RNG seed programmed by the OEM boot module and program it to flash. This process must occur on every reset.
- Execute the `CREATE_SUBKEY_PI` to create a new subkey that is used for protecting the Marvell® WTM state and write the resulting subkey into flash.
- Execute the `SAVE_WTM_STATE_PI` and write the resulting state data into flash memory at the designated address.
- Program the platform configuration fuses, if not done at manufacturing.

The Marvell® WTM driver must generate a new RNG seed using the Marvell® WTM for seeding into the Marvell® WTM on the next subsequent boot, and program the new seed to flash. The Marvell® WTM state in flash must also be updated before the platform is put to sleep.

4 Methods for Platform Provisioning

The requirements for platform provisioning depend on the usage model selected. This chapter provides some guidance on provisioning a platform for operation with the Boot ROM. “Provisioning a platform” means performing the required steps to turn an uninitialized system into an initialized system capable of booting to an operating system. Differences between a development system and a manufacturing system are highlighted whenever possible.

4.1 Non-Trusted Provisioning

The first step toward provisioning a non-trusted system is to review the use cases in Section 8.1, “Operation with a Non-Trusted Image Module” to determine the requirements. During this time, consider the following:

1. How the flash device is programmed. Several options are available:
   - Programmed via the JTAG port using the JTAG software package
   - Preprogrammed by a flash programming vendor
   - Programmed using a separate software image that is downloaded over one of the available ports
2. The flash device that is used for booting the system. The options are:
   - X16 NAND device on data flash controller Chip Select 0
   - X8 NAND device on data flash controller Chip Select 0
   - XIP device on the static memory controller Chip Select 2
3. The size of the first boot loader binary. Size implications must be reviewed when using a NAND flash device. For larger boot loaders, the use of a non-trusted image module and the bad-block management is required.
4. The level of verification that is required. The use of the Non-Trusted Image Module allows for a cyclic redundancy check (CRC) to be performed on the image.
5. Whether the same OEM boot module is used for both trusted and non-trusted systems. If both trusted and non-trusted platforms are supported, use the non-trusted image module. This selection allows one OEM boot module binary to be used on both platforms.

4.1.1 Provisioning an Unprogrammed Non-Trusted Boot Platform Using a Device Keying Binary

Complete these steps fully to provision an unprogrammed platform using a Device Keying Binary. Troubleshooting a secure enabled processor is more complicated because the JTAG port is disabled by default.

1. Decide on the usage model for booting the system. See Section 8.2, “Operation Without a Non-Trusted Image Module”.
2. Prepare a non-trusted image module binary and Device Keying Binary using the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
3. Prepare the non-trusted image module binary, OEM boot module, and associated operating system images using the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
   a) Boot the target platform and first download the non-trusted image module and associated Device Keying Binary created in Step 2 using the download tool available in the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.

b) The Device Keying Binary runs on the system and must perform all of the requirements documented in Section 4.1.1.1.
c) The non-trusted image module, OEM boot module, and associated OS images created in Step 3 are downloaded by the Device Keying Binary using the download tool available in the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
d) The Device Keying Binary must have the built-in capabilities to allow debug and testing of the non-trusted image module, OEM boot module, and associated OS images created in Step 3

4. As a last step, the non-trusted boot operation should be verified from a power-on reset.

The non-trusted boot process occurs on the platform upon every reset of an initialized platform. The non-trusted boot processes use the information stored in the non-trusted image module to load the images from flash memory before transferring control, if required.

4.1.1.1 Device Keying Binary Requirements for an Unprogrammed Non-Trusted System

The Device Keying Binary is responsible for provisioning and preparing an uninitialized system for initial boot. It must determine the flash used to boot, program the proper images to the flash, and if the platform is a NAND platform, validate or create the relocation table. In addition, an OEM may want to create multiple versions of the Device Keying Binary, one for use in manufacturing and one for use in development. The development Device Keying Binary could be used to aid in platform debugging.

Marvell provides the Marvell® Wireless Trusted Platform Tool Package as an example for OEMs. This package contains all of the host tools and middleware required for both trusted and traditional boot. Contact your local Marvell field application engineer for more information.

The Device Keying Binary is responsible for completing the following on non-trusted boot platforms:

- Provide an interface through the UART or USB port to print messages and download binary images.
- Set up the DDR memory and all necessary flashes to store the downloaded images. At a minimum, this would include the OEM boot module.
- Create the initial bad-block table, if the flash signature in the Non-Trusted Image Module indicates a setup for NAND.
- Perform a checksum on the images against the values stored in the non-trusted image module to validate a correct download.

4.2 Trusted Provisioning

The first step towards provisioning a trusted system is to review the usage cases in Section 9.1, “Trusted Boot Usage Cases” and determine the requirements. When doing so, consider the following:

1. How the flash device is programmed. Several options are available:
   a) A separate software image that is downloaded over one of the available ports.
   b) A system that is preprogrammed by a flash programming vendor.
2. The flash device that is used for booting the system. The options are:
   a) X16 NAND device on data flash controller Chip Select 0.
   b) X8 NAND device on data flash controller Chip Select 0.
   c) XIP device on the static memory controller Chip Select 2.
3. The Device Keying Binary that is used. Marvell provides a sample Device Keying Binary that performs all of the necessary tasks required to provision a trusted system. If system-level debug is also required, modification of the Device Keying Binary may be necessary or a new Device Keying Binary can be developed by the OEM.

4. If the same OEM boot module is used for both trusted and non-trusted systems. If both trusted and non-trusted platforms are supported, use the Non-Trusted Image Module for the non-trusted system, which allows one OEM boot module binary to be used on both platforms.Review the requirements for the trusted OEM boot module in Chapter 9, “Trusted Boot Operation”.
5. The tools that are used to generate the Trusted Image Module. Marvell provides a sample tool for generating the Trusted Image Module described in Chapter 7, “Trusted Image Module”. An OEM may also generate a separate tool.
6. How the system is debugged and how errors are diagnosed. The JTAG port on a trusted boot processor is disabled by default, which means different strategies must be deployed for troubleshooting. Refer to Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 to identify processors enabled for trusted boot support.

4.2.1 Provisioning an Unprogrammed Trusted Boot Platform Using a Device Keying Binary

Complete the following steps to provision an unprogrammed platform using a Device Keying Binary. On a secure enabled processor, the JTAG port is disabled by default and makes troubleshooting more complicated.

1. Decide on the usage model for booting the system, according to those documented in Chapter 9, “Trusted Boot Operation”.
2. Prepare a trusted image module binary and Device Keying Binary using the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
3. Prepare the Trusted Image Module binary, the OEM boot module, and the associated operating system images using the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
4. Boot the target platform and first download the trusted image module and associated Device Keying Binary created in Step 2 using the download tool available in the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
5. The Device Keying Binary runs on the system and must perform all of the requirements documented in Section 4.2.2, “Device Keying Process”.
6. The Trusted Image Module, OEM boot module, and associated operating system images created in Step 3 are downloaded by the Device Keying Binary using the download tool available in the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the OEM.
7. The Device Keying Binary must have built-in capabilities to allow debug and testing of the Trusted Image Module, OEM boot module, and associated operating system images created in Step 3. This capability can be accomplished by allowing JTAG re-enabling or providing debug and test functionality over the download port.
8. As a last step, the trusted boot operation should be verified from a power-on reset.

Warning

Warning

Once the fuses are programmed on the processor, they cannot be changed. Verify correct boot operation before programming all of the fuses to avoid incorrect configuration.

The device keying process is initiated by the Boot ROM when the platform fuses indicate the platform is in the uninitialized state. A Device Keying Binary is used to load images into the boot flash and set other security-related information on the processor with XScale® technology. Once the provision is successful and the fuses are programmed, the platform becomes an initialized platform.

The trusted boot process occurs on the platform upon every reset of an initialized platform. The trusted boot processes use the security information stored in the trusted image module and the security information programmed by the Device Keying Binary to integrity check the images loaded from the flash before transferring control. If the validation process fails, the boot operation is halted.

The validation process is a series of checks on the Marvell® WTM and the images loaded into the flash. If the Marvell® WTM is successfully initialized, the Boot ROM starts the validation process on the images in the flash. Validation uses the RSA digital signature and the SHA-1 based digital signature verification of the trusted image module and OEM boot module in the validation process. Computed hashes are compared with hashes stored in the flash, which allows the Boot ROM to detect any changes that have occurred since the last boot attempt. If an error occurs at any point in the process, the boot process halts.

4.2.2 Device Keying Process

The device keying process occurs on an uninitialized platform, as defined by the Boot ROM fuses. Device keying is performed during platform manufacturing, when the platform configuration fuses can be programmed through the Marvell® WTM. The fuses can also be programmed during the manufacturing of the processors at the Marvell factory. The OEM must use the Marvell® Wireless Trusted Platform Tool Package to generate a trusted image module with all of the necessary security information for the trusted boot process. Optionally, an OEM could develop custom tools to generate the trusted image module according to Chapter 7, “Trusted Image Module”.

1. During the device keying process, the Boot ROM listens for a download request on either the UART, differential USB, or single-ended USB port to start the download process.
2. If a request has been initiated, the Boot ROM begins by downloading a trusted image module to a reserved location in the internal SRAM. This trusted image module should cover the Device Keying Binary that is downloaded next.
3. After examining the load address for the Device Keying Binary from the trusted image module, the Boot ROM downloads the Device Keying Binary to this address and the Device Keying Binary should have been linked to execute from this location. Depending on the setting of the secure download enable fuse, the trusted image module and the Device Keying Binary may be integrity checked or control is transferred to the Device Keying Binary directly.
4. Once the downloaded Device Keying Binary is given control, it is responsible for completing the requirements listed in Section 4.2.2.1, “Device Keying Binary Requirements for an Unprogrammed System”. The OEM should have its own proprietary secure scheme implemented in the downloaded OEM boot image. Execution of this scheme allows further download and future security checking of the OEM’s supplementary binaries for its OS/application/data, as well as the mobile operator’s service provisioning modules.

To allow an OEM to debug images, hooks have been provided in the Boot ROM to allow commands to be issued. Refer to the command protocol section in Chapter 11, “Communication Protocol” for details about the commands. The debug commands are allowed only on an uninitialized platform. Once the platform fuses are configured, debug commands are disabled. The Device Keying Binary must allow for the debug capabilities to be used by having an option to skip programming the fuses, which also implies that the Device Keying Binary should have a mode where it skips the download of images and only programs the fuses. The way this process is implemented as defined by the OEM. One suggestion is to support additional commands through the port protocol to allow for separating the steps of the platform provisioning.

4.2.2.1 Device Keying Binary Requirements for an Unprogrammed System

The Device Keying Binary is responsible for provisioning and preparing an uninitialized system for initial boot. It must determine the flash that is used to boot, program the proper images to the flash, integrity check images if required (secure download enabled), program the platform configuration fuses, program the one-time programmable registers on NOR platforms, and if the platform is a NAND platform, validate or create the relocation table. In addition, an OEM may want to create

multiple versions of the Device Keying Binary, one for use in manufacturing and one for use in development. The development Device Keying Binary could be used to aid in debugging the platform.

Marvell provides the Marvell® Wireless Trusted Platform Tool Package as an example for OEMs. This package contains all of the host tools and middleware required for both trusted and traditional boot. Contact your local Marvell field applications engineer for more information.

The Device Keying Binary is responsible for completing the following on trusted boot platforms:

- Provide an interface through the UART or USB port to print messages, when run-time progress and error reporting are needed, and download binary images.
- Set up the DDR memory and all necessary flashes to store the downloaded images. At a minimum, this includes the OEM boot module.
- If the flash signature in the trusted image module indicates a setup for NAND, create the initial bad block table.
- Integrity check the images against the trusted image module.
- Find a good entropy source to generate/collect a five-word random number generator (RNG) seed and program it into flash where the Marvell® WTM save state is newly created, according to the trusted image module.
- Program the encrypted hash value of the OEM platform verification key into the flash one-time programmable Register 0 (64-bits) and the remaining 96 bits into the lower half of Register 1 (for platforms using XIP-based flashes only).
- Program the hash value of the JTAG re-enabling key, also referred to as the corrupted OEM boot module reverification key, into the upper half of Register 1 (32 bits) and the remaining 128 bits into Register 2 and lock Registers 0, 1, and 2 respectively (for platforms using XIP-based flashes only).
- Program the OEM wrapped verification keys (24 bits for A0, 48 bits for stepping B0 and forwards) into the Marvell® WTM fuses using the `OEM_Platform_Bind_PI` via an `ippCP` call.
- Program the platform configuration fuses in the Marvell® Wireless Trusted Module.

Figure 5: Device Keying Binary Requirements Flow for an Unprogrammed System

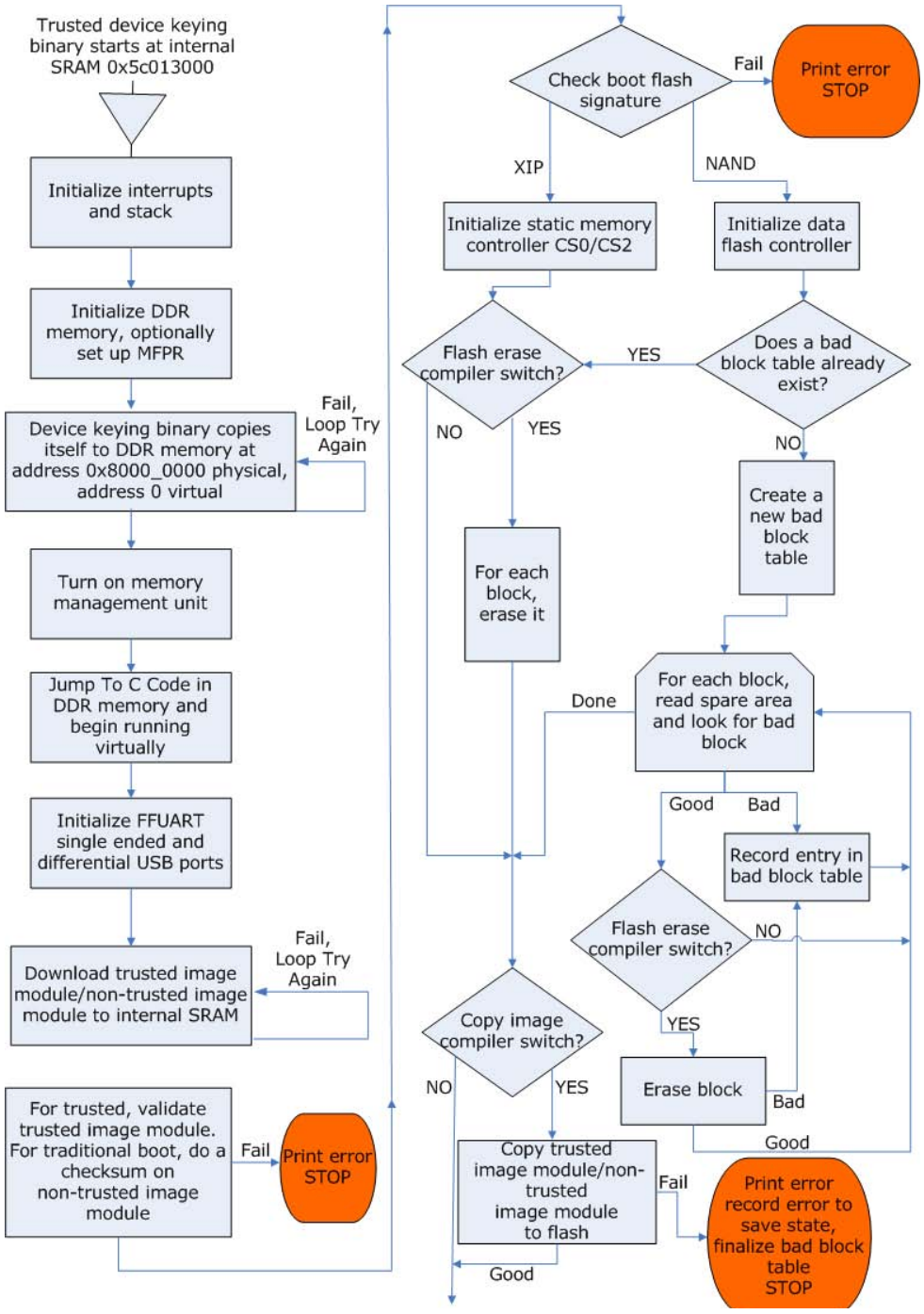
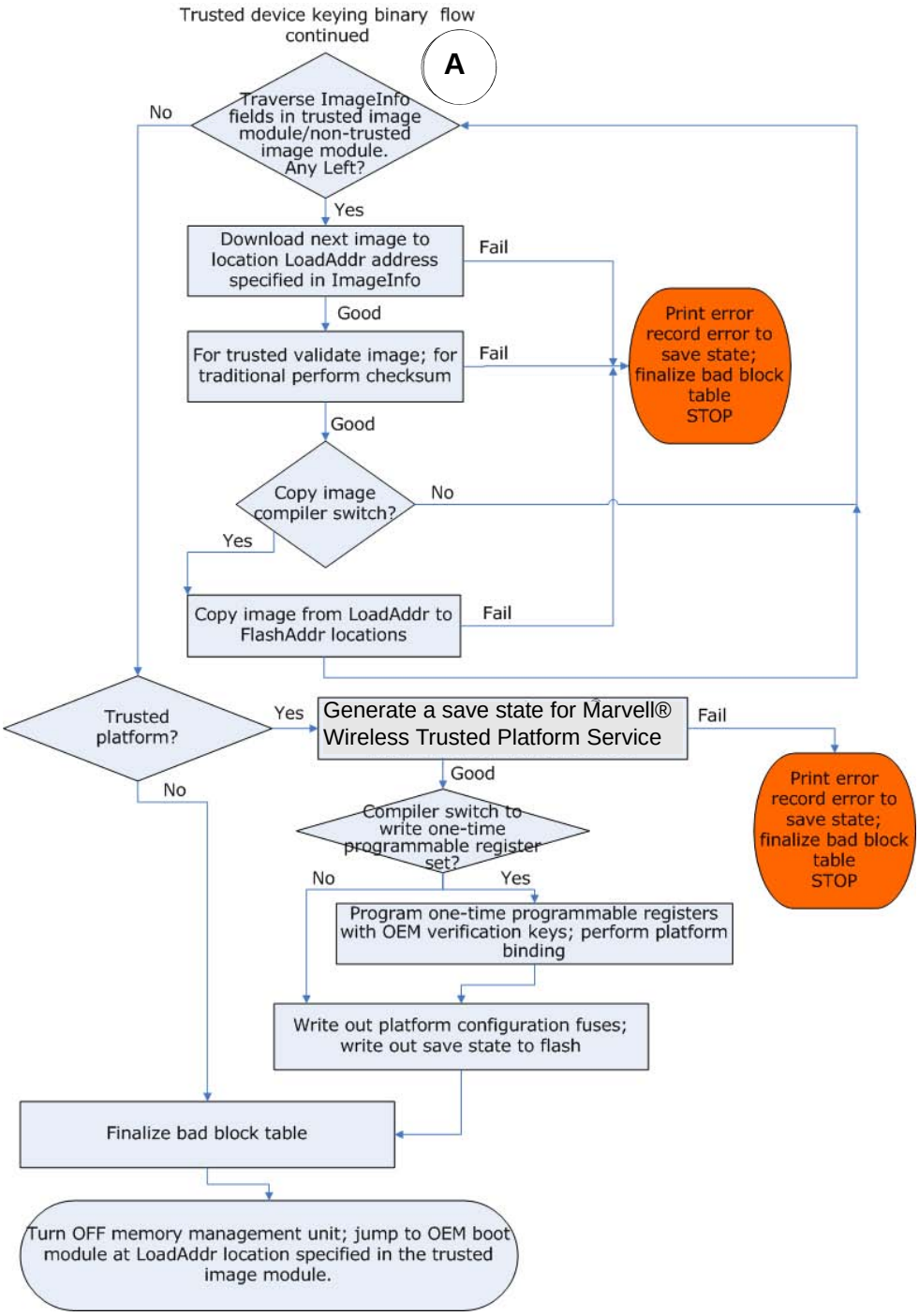


Figure 6: Device Keying Binary Requirements Flow for an Unprogrammed System (cont)



4.2.2.2 Size Restrictions for the Device Keying Binary

The size of the Device Keying Binary should be restricted, based on the available internal SRAM on the processor. Refer to the internal SRAM usage in Section 5.7, “Internal SRAM Usage”. Marvell suggests restricting the Device Keying Binary size to less than 64 KB, because a minimum of 128 KB of internal SRAM is used.

5 Boot ROM Implementation Details

The following sections detail the Boot ROM address maps.

5.1 Non-Trusted Boot Address Map

For non-trusted platforms, control is handed to the OEM boot module without security validation. For consistency with trusted platforms, the use of an image module is supported to provide image information in the same manner as a trusted platform. A Non-Trusted Image Module also allows for a CRC to be performed on an image before control is transferred. See Chapter 6, “Non-Trusted Image Module” for more details.

If a Non-Trusted Image Module is used, it must be loaded at the location shown in Table 7, “Non-Trusted Image Module Locations". NTIM headers are only supported in version 3.xx of the Boot ROM.

If the Non-Trusted Image Module is not used, then the binary OEM boot module image must also be loaded at the location also shown in Table 8 “OEM Boot Module (OBM) locations when No NTIM is used”.

For NAND platforms, if the Non-Trusted Image Module is not used, the OEM boot module image is restricted in size. See Section 6.1.2, “Flash Information” for more details.

Table 7: Non-Trusted Image Module Locations

Boot Device	Location in the Device	Runtime Location of the OEM Boot Module
Chip Select 0 (nCS0)*	Block 0 offset 0xC000	Based on NTIM
Chip Select 2 (nCS2)	Block 0 Offset 0x0	Based on NTIM
NAND Device on ND_nCS0	Block 0 offset 0x0	Based on NTIM
OneNAND	Block 0 offset 0x0	Based on NTIM
MDOC MSys	Partition 2 offset 0x0	Based on NTIM

Table 8: OEM Boot Module (OBM) locations when No NTIM is used

Boot Device	Location in the Device	Runtime Location of the OEM Boot Module
Chip Select 0 (nCS0)*	Block 0 offset 0xC000	0x0000__C000
Chip Select 2 (nCS2)	Block 0 Offset 0x0	0x1000_0000
NAND Device on ND_nCS0	Block 0 offset 0x8 - version 2.xx Block 0 offset 0x0 - version 3.xx	0x5C01_4000 - version 2.xx 0x5C01_3000 - version 3.xx
OneNAND	XIP Area	0x1000_0000
MDOC MSys	XIP Area	0x1000_0000

*NOT Managed NAND (OneNand + mDOC), only NOR/XIP Flash

5.1.1 Trusted Boot Address Map

Trusted platforms require the use of the trusted image module. The trusted image module must be loaded at the locations shown in Table 9, based on the flash device used. The trusted image module must be created using the Marvell® Wireless Trusted Platform Tool Package image building tools, or an OEM equivalent. See Chapter 7, "Trusted Image Module," for details on the trusted image module format. Contact your Field Application Engineer for information on the Marvell® Wireless Trusted Platform Tool Package.

Table 9: Trusted Image Module Locations for Trusted Boot

Boot Device	Location in the Device	Runtime Location
Chip Select 0 (nCS0)	Block 0 offset 0xC000	Based on TIM
Chip Select 2 (nCS2)	Block 0 Offset 0x0	Based on TIM
NAND Device on ND_nCS0	Block 0 offset 0x0	Based on TIM
OneNAND	Block 0 offset 0x0	Based on TIM
MDOC MSys	Partition 2 offset 0x0	Based on TIM

5.1.2 NTIM/TIM Execution

The NTIM/TIM headers are used by the processor and Boot ROM to coordinate the image boot process. Each of the binary images that are specified in the headers is loaded and executed in turn. The last image specified has the Next Image ID marker as: 0xFFF_FFFF which informs the Boot ROM that there are no more images to load.

The header is first loaded to 0x5C00_8000 and contains all the image details. The next image to load is the OBM followed by the OS boot image or OEM-specific image. If any images must be loaded into DDR memory space, then the OBM image must provide the capability of configuring all necessary GPIOs and interface controllers such as the Dynamic Memory Controller.

If the OBM does not configure the DDR, then the Boot ROM will fail the attempt to load the OS image and the boot process will fail. The following is an example of an NTIM header, which is typical for a Marvell Board Support Package (BSP). This example also has a GPIO package.

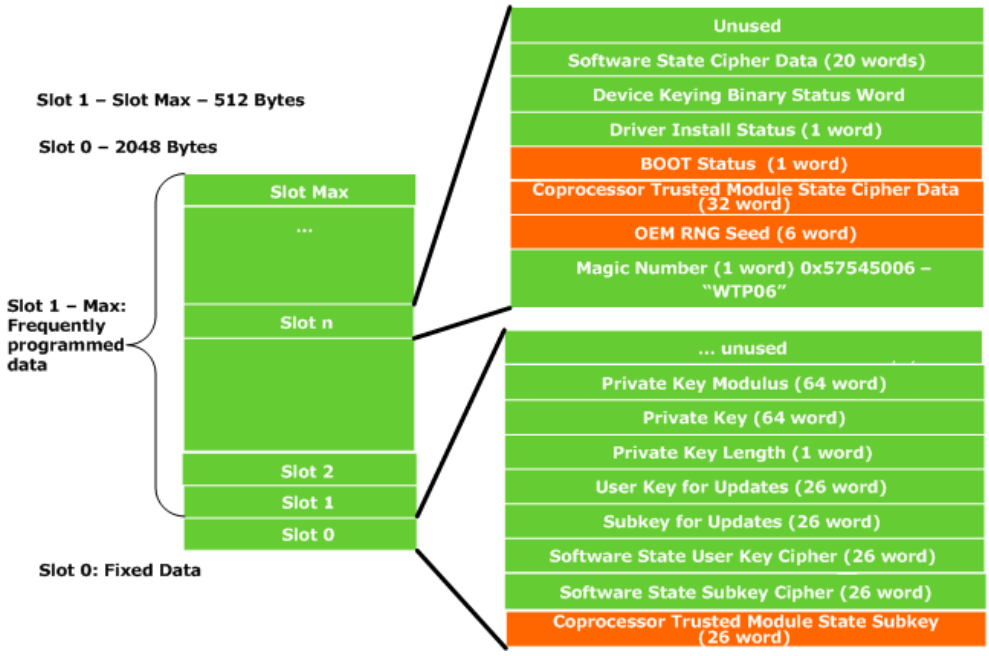
```
Version: 0x030101
Trusted: 0
Issue Date: 0x01252006
OEM UniqueID: 0xfedcba98
Boot Flash Signature: 0x4e414e04
Number of Images: 3
Size of Reserved in bytes: 0


Image ID: 0x54494D48
Next Image ID: 0x4F424D49
Flash Entry Address: 0x0
Load Address: 0x5c008000
Image Size To CRC in bytes: 0xff
Image Filename: NTIM_LV.bin
```

```
Image ID: 0x4F424D49
Next Image ID: 0x4F534c4F
Flash Entry Address: 0x20000
Load Address: 0x5c013000
Image Size To CRC in bytes: 0
Image Filename: MHLV_wince_NTOBM.bin

Image ID: 0x4F534c4F
Next Image ID: 0xFFFFFFFF
Flash Entry Address: 0x40000
Load Address: 0x83C00000
Image Size To CRC in bytes: 0
Image Filename: eboot.nb0

Reserved Data:
0x4F505448 // Package structure identifier
0x00000002 // number of packages, note must include termination package

0x4750494f // GPIO package
0x00000024 // package size
0x00000003 // number of pairs
0x40e104d0 // MFPR reg address
0x00000840 // data
0x40e00014 // GPDR2 reg address
0x00001000 // set GPIO 76 as output
0x40e00020 // GPSR2 reg address
0x00001000 // set GPIO 76 high

0x5465726D // Termination package
0x00000008
```

5.1.3 Requirements for the Wireless Trusted Module Save State

The Marvell® Wireless Trusted Module (Marvell® WTM) save state is implemented with a slot-based mechanism to provide wear leveling on the flash device. Figure 7 shows the layout for the Marvell® WTM save state implementation. Two blocks of flash are currently allocated for the Marvell® WTM save state: a primary Marvell® WTM save state, and a backup Marvell® WTM save state. The location of the Marvell® WTM state files is part of the information located in the trusted image module and covered by the digital signature to prevent attacks on the system.

Figure 7: Coprocessor Trusted Module Save State Implementation



5.1.4 Specific Requirements for NAND Platforms

A NAND platform requires support for bad-block management, as well as error detection and correction. `ECC_EN` and `SPARE_EN` are enabled when programming NAND blocks using the NAND Flash Controller. The Boot ROM makes use of the Marvell bad-block management scheme if the bad-block table is present. If the Marvell bad-block table is not present, the OEM boot module is limited in size as defined in Table 10.

Table 10: OEM Boot Module Sizes Without Marvell Bad Block Management

Platform Configuration	Small Block NAND	Large Block NAND
Non-Trusted	Block 0 – 1 page (15.5 KB)	Block 0 -1 page (127 KB)
Non-Trusted	Block 0 – image module - 1 page (approximately 15 KB)	Block 0 – image module -1 page (approximately 126 KB)
Trusted	Block 0 – Trusted Image Module size – 1 page (11.5 KB)	Block 0 – Trusted Image Module size – 1 page (123 KB)

The trusted size allows for the trusted image module size to be a maximum of 4 KB. If the image module is also used for a non-trusted platform, one page is reserved for the image module.

If the Marvell bad-block scheme is implemented and the image module or trusted image module is implemented, the OEM boot module size restriction does not exist and the OEM boot module can be any size. The size of the OEM boot module is determined from the image module or trusted image module as well as the starting location. The OEM boot module likely consumes contiguous blocks in the NAND device, (Blocks 1 through 3, for example). The image cannot be broken into non-contiguous blocks unless a block is relocated through the bad-block table. Refer to Section 3 "Software Requirements" for more details on NAND bad-block management.

Note

Note The Marvell bad-block scheme is code used for NAND bad-block management. This code is available as example code in the OBM source files. Contact your local FAE for BSP availability.

5.2 Boot ROM NAND Device Support

The Boot ROM for the PXA3xx Processor Family and Tavor Processor supports many different SLC NAND devices such as:

- Large Block NAND x8 and x16
- Small Block NAND x16

5.2.1 Boot ROM NAND Device Recognition

An algorithm has been implemented to enable booting from different types of NAND devices. The algorithm works as follows:

1. Boot ROM issues the reset command 0xFF

Note

Note This reset command is required as part of the ONFI specification. NAND devices that require the reset command are only supported natively by V3.32 Boot ROM and later. See Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 for overview of processors that support this feature.

2. The Boot ROM issues a read_id command to the device retrieving two bytes of data.
3. The two bytes (manufacturer ID and device ID) are compared against the small block codes in Table 11. All Boot ROM versions support these devices.
   - If the device is a small block device then the Boot ROM configures the NAND controller for small block operation with a 512-Byte page size and 16 KB block size.

Table 11: Small Block Devices

Manufacturer	Manufacturer Code	Device Codes
Samsung	0xEC	0x71, 0x78, 0x79, 0x72, 0x74, 0x36, 0x76, 0x46, 0x56, 0x35, 0x75, 0x45, 0x55, 0x33, 0x73, 0x43, 0x53, 0x39, 0xE6, 0x49, 0x59
Toshiba	0x98	0x46, 0x79, 0x75, 0x73, 0x72, 0xE6
Hynix	0xAD	0x76, 0x56, 0x36, 0x46, 0x75, 0x55, 0x35, 0x45, 0x73, 0x53, 0x49
ST Micro	0x20	0x73, 0x35, 0x75, 0x45, 0x55, 0x76, 0x36, 0x46, 0x56, 0x79, 0x39, 0x49, 0x59

4. For unknown small-block devices, the Boot ROM issues another read_id retrieving the 4th byte of data from the device.
   - The Boot ROM uses the information returned in the 4th byte to determine the page and block size of the NAND device. Bits 1 and 0 for the page size and bits 5 and 4 for the block size. Refer to Table 5, Version 3.xx OBM usage, on page 19.
   - The Boot ROM does NOT use the manufacturer and device codes for Large Block NAND configuration. The NFC is configured for large block based on the device parameters.
   - The NFC has a limit of 2048Byte page size.
5. The Boot ROM then configures the command set for the appropriate NAND device (based on the above steps) and continues with normal read operation.

Note

Note

This was originally documented as the Boot ROM expecting 0x15 for x8 large block NAND and 0x55 for x16 large block NAND devices. Although the entire 4th byte is read from the NAND device, the Boot ROM uses only bits 0, 1, 4, and 5, not the entire byte (all 8 bits) to configure the appropriate memory device.

Figure 8: Electronic Signature Requirements

		Electronic Signature Requirements							
		Block Size		Block Size Decoded			Page Size		Page Size Decoder
IO7	IO6	IO5	IO4		IO3	IO2	IO1	IO0	
		0	0	64K			0	0	Not Supported
		0	1	128K			0	1	2K
		1	0	Not Supported			1	0	Not Supported
		1	1	Not Supported			1	1	Not Supported

The NAND Flash Controller (NFC) is set up for the appropriate NAND configuration as noted in Table 12, "NAND Flash Controller Initial Register Settings". NDCR[DWIDTH_C] and NDCR[DWIDTH_M] are configured by the Boot ROM depending on the boot configuration SKU of the processor.

Table 12: NAND Flash Controller Initial Register Settings

Register	Value for Small Block Operation	Value for Large Block Operation
NDCR[DWIDTH_C] and NDCR[DWIDTH_M] (bits27:26) initial settings are determined by the boot state fuses	0xCC02_1FFF	0xCD04_1FFF
Timing register 0	0x003F_3F3F	0x003F_3F3F
Timing register 1	0x1FF0_C0FF	0x1FF0_C0FF

Table 13: NAND Command Set

Command	Small Block Command Code	Large Block Command Code
Read	0x0000_	0x3000
Read Status	0x0070	0x0070
Read ID	0x0090	0x0090

Note

Note

Some NAND devices power up with the blocks in a locked state. To ensure the blocks can be unlocked with software, refer to the NFC chapter of the *PXA3xx Processor Family Vol. II: Memory Controller Configuration Developers Manual*.

5.3 XIP Flash Support

The processor Boot ROM supports NOR Flash memory on the data flash interface (DFI) bus. Flash is supported via a command set, not through a particular JEDEC ID. See Table 14 for supported commands.

Any device is supported provided that the device is compliant with the commands as described in Table 14.

The processor can natively support AA/D muxed memories. Other memories may be connected and booted from but external latches are required. Refer to the *PXA3xx Processor Family Developers Manual* for more details on the AA/D muxed bus.

NOR like NAND devices such as M-Systems mDOC and Samsung One NAND are also supported via the DFI bus. Later versions of Boot ROM have integrated the driver to these managed NAND devices while other versions of Boot ROM supports these devices using the XIP area. Refer to Chapter 5, “OneNAND Support” and Chapter 5, “mDOC Support” for more details.

Table 14: Flash Commands Supported by the Boot ROM

Flash Command Name	Flash Command Data	Flash Type
Read Array	0xFF	Intel StrataFlash® Wireless Memory (XIPA), Intel StrataFlash® Cellular Memory (M18) (XIPB)
Read Device Identifier	0x90	Intel StrataFlash® Wireless Memory (XIPA), Intel StrataFlash® Cellular Memory (M18) (XIPB)
Clear Status Register	0x50	Intel StrataFlash® Wireless Memory (XIPA), Intel StrataFlash® Cellular Memory (M18) (XIPB)
Word Program	0x40	Intel StrataFlash® Wireless Memory (XIPA)
Word Program (Intel StrataFlash® Cellular Memory (M18))	0x41	Intel StrataFlash® Cellular Memory (M18) (XIPB)
Unlock Block	0x60/0xD0	Intel StrataFlash® Wireless Memory (XIPA), Intel StrataFlash® Cellular Memory (M18) (XIPB)
NOTE: The command set supported by the Boot ROM is not specific to Intel StrataFlash®. Any NOR flash device is supported if that device supports the same command sets as described in this table. **NOTE:** XIPA and XIPB are used for reference in other Boot ROM chapters.		

5.3.1 NOR Flash One-Time Programmable Register Usage

The NOR Flash memories have one time programmable (OTP) register bits that may be used with the PXA3xx processor.

The OEM public key hash that protects the OEM boot module is stored in one-time programmable Registers 0 and 1. The JTAG key hash is stored in Registers 1 and 2. There are 16 registers in the one-time programmable section, each is 128 bits. Register 0 comes with a flash ID preprogrammed in the bottom 64 bits; the upper 64 bits are available for general usage.

The 64 bits of the OEM public key hash must be stored in the upper half of Register 0, with the remaining 96 bits in Register 1.

The 32 bits of the OEM JTAG key hash must be stored in the upper half of Register 1, with the remaining 128 bits in Register 2. Registers 0, 1, and 2 must be locked after they are programmed.

5.4 Managed NAND Memory Support

When the Boot ROM is defined to have support for OneNAND and mDOC memory devices (refer to Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15), this basically means

that the Boot ROM has knowledge of the memory system. There are specific requirements for accessing these memories and also the location of certain boot images.

The Boot ROM can program the static memory controller and can directly access the correct memory locations to execute the OBM images and NTIM/TIM headers.

Any Boot ROM that does not natively support mDOC and OneNAND can still use these memories; however, the XIP areas of these devices are the only memory locations that the Boot ROM accesses.

Even though the Boot ROM may have integrated drivers for these memories, it does not replace the necessity for OS-level flash drivers. These drivers must be implemented in the OS.

The following two sections discuss some detail regarding support of OneNAND and mDOC.

The appropriate BootFlashSign definition must be used to define what memory is to be booted from. These definitions are defined in Table 37, BootFlashSign Definitions, on page 65 for non-trusted boot and Section 5.4, “Managed NAND Memory Support” on page 42 for trusted boot.

5.5 OneNAND Support

The Samsung OneNAND memory is interfaced to the processor using the DFI bus with nCS2 as the boot chip select.

The location of the NTIM or TIM header is address 0x0, which applies to Boot ROM versions 3.32 and later. This address should be specified in the header itself and the header binary must reside at this location. The OBM location is also defined in the header and must be loaded at that location.

Note

Note

When connecting the OneNAND memory to a processor with a Boot ROM version 3.27 or earlier, the NTIM/TIM must be loaded to the XIP area at address 0x1000_0000. The OBM image also resides in the XIP space and must contain a OneNAND driver. The Boot ROM cannot directly access the OneNAND memory until the driver has been loaded from the OBM boot code.

Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 states which version of Boot ROM natively supports this memory device.

Table 8, OEM Boot Module (OBM) locations when No NTIM is used, on page 35 states which version of Boot ROM natively supports this memory device; therefore, this list is only applicable to those processors. This list specifies the device IDs and accompanying stepping generation codes that are supported natively by those processors/Boot ROMs.

Not only must the Manufacturing ID of 0xEC be read for these versions of Boot ROM, but also the Device and Generation IDs must be matched with the details in Table 10.

For all subsequent Boot ROM versions, only the Manufacturer ID of 0xEC must be read. The Boot ROM then reads the Device and Generation IDs to calculate the density of the OneNAND without having them predefined as with previous generations.

To summarize the requirements:

- Boot ROM versions 3.32 only work with the devices listed in Table 10
- All subsequent Boot ROM versions 3.33 and future require only the Manufacturer ID of 0xEC. This is a mandatory requirement due to accessing the OneNAND register definitions and layout.

Note

Version 3.xx Boot ROMs have the backwards compatibility mode where the header is optional; however, if using OneNAND in the system, a header (NTIM or TIM) MUST be implemented (applicable to all versions of Boot ROM).

5.5.1 Exiting Low Power Mode and Resets with OneNAND

Resuming from low power modes and reset exits are different depending on whether the Boot ROM has the integrated driver. For example, the PXA320 B2 does not have the integrated driver for OneNAND memory. The Boot ROM can access only the XIP area. The PXA310 A2 does have the driver embedded so the Boot ROM enables direct accesses to the OneNAND memory device at boot time, reset, and low-power mode exits as described in Table 9, Trusted Image Module Locations for Trusted Boot, on page 36.

Table 15: OneNAND Device ID Support

Processor Stepping and Boot ROM Version	Samsung OneNAND Device Information		
	Device ID	Generation ID	Description
All Devices; PXA302 & PXA312 (PoP) Tavor B0.b Version 3.32 PXA31x A2 Version 3.33	0x30	0	1G Mux
	0x31	0	1G Mux
	0x30	1	1G Mux A-die
	0x31	1	1G Mux A-die
	0x20	0	512M Mux
	0x21	0	512M Mux
	0x20	2	512M Mux B-die
	0x21	2	512M Mux B-die
	0x38	0	1G DDP Mux
	0x39	0	1G DDP Mux
	0x48	0	2G DDP Mux
	0x49	0	2G DDP Mux
	0x10	0	256M Mux
	0x10	1	256M Mux
	0x11	0	256M Mux
	0x11	1	256M Mux
	0x00	0	128M Mux
	0x01	0	128M Mux
	0x40	0	2G Mux M-die
	0x41	0	2G Mux M-die
PXA31x A2 Version 3.33	0x58	0	4G Mux M-die

5.6 mDOC Support

The Sandisk mDOC memory is interfaced to the processor using the DFI bus with nCS2 as the boot chip select, much the same as OneNAND.

Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 states which version of Boot ROM natively supports this memory device.

The mDOC device acts as a NAND disk; however, no file system is mounted during the processor boot operation. Instead, the mDOC driver code resident in the processor Boot ROM allows accesses to the mDOC device via formatted partitions and sector offsets within the partitions.

The TIM/NTIM must reside in Partition 2, Sector Offset 0. Any other images (OBM) must reside in Partition 2 or greater (applicable to Boot ROM versions 3.32 and later).

Note

Note

When connecting the mDOC memory to a processor with a Boot ROM version 3.27 or earlier, the NTIM/TIM must be loaded to the XIP area at address 0x1000_0000. The OBM image also resides in the XIP space and must contain an mDOC driver. The Boot ROM cannot directly access the mDOC memory until the driver has been loaded from the OBM boot code.

Partition 1 is an OTP partition that stores keys for trusted boot operations, but otherwise is not usable. mDOC sectors are 512 bytes.

The 32-bit flash address for mDOC images are calculated as follows:

28 bit sector offset	4 bit partition #

Here are examples of how to calculate Flash positions within the mDOC device for use with input files for the TBB tool:

	mDOC address

NTIM (must be in Partition 2, Sector Offset 0)	= 0x2
OBM in Partition 5 starting at Sector Offset 32	= 0x205
OBM in Partition 3 starting at Sector Offset 506	= 0x1FA3

Note

Note

Version 3.xx Boot ROMs have the backwards compatibility mode where the header is optional; however, if using mDOC in the system, a header (NTIM or TIM) MUST be implemented (applies to ALL versions of Boot ROM).

5.6.1 Exiting Low Power Mode and Resets with mDOC

Resuming from low power modes and reset exits are different depending on whether the Boot ROM has the integrated driver. For example, the PXA320 B2 does not have the integrated driver for mDOC memory. The Boot ROM can access only the XIP area. The PXA310 A2 does have the driver embedded so the Boot ROM enables direct accesses to the mDOC memory device at boot time, reset and low power mode exits as described in Table 9, Trusted Image Module Locations for Trusted Boot, on page 36.

5.7 Internal SRAM Usage

The internal SRAM is used as a data area and to download images from the download ports or the NAND flash. The internal SRAM starts at `0x5c00_0000` and the end is implementation defined (up to 768KBytes max for PXA32x).

By default, the lower 32KB of internal SRAM is locked by the Wireless Trusted Module (WTM). The Boot ROM unlocks 31KB of SRAM from 0x5c00_0400 to 0x5c00_8000 and uses the space to set up stack usage.

Note

Note

Software must NOT disable the D0CKEN_B[17] bit prior to entering low power modes. Although this bit allows software to enable and disable the Mini-LCD clock, this clock is also used internally during this sequence. Refer to the *PXA3xx Processor Family Developers Manual* for details on clock gating.

The Boot ROM uses 44 KB (ver 3.xx) or 48 KB (ver 2.xx) above the 32KB for the data area. All image downloads are to the addresses, depending on the version of Boot ROM. Refer to Table 3, “Version 3.xx ISRAM Usage" and Table 4, “Version 2.xx OBM Usage" for details.

5.8 Handling Power Mode and Reset Transitions

The Boot ROM manages the task of booting up the system which is not only at power-on-reset but also reset exits. There are differences that must be managed, depending on the mode or reset state that is being entered or exited and the version of Boot ROM. This section establishes those differences so that implementation into a system design is better understood for both hardware and software perspectives.

The Boot ROM performs the basic checks to determine the last reset exit or power mode exit.

The OBM image must be able to successfully transition the processor through the entire low power or reset, entry, and exit sequence.

Any exit from the various resets or low power mode exits, including booting the processor, requires steps that must be followed. These requirements are documented in more detail in the processor-specific developer manuals. Marvell recommends that these chapters be consulted to ensure proper system reset procedures and low power modes resumes.

The Boot ROM follows basic tasks as described in Table 15, OneNAND Device ID Support, on page 44.

Table 16: Overview of Resets and Power Modes

Power Mode or Reset Exit	V2.xx Boot ROM	V3.xx Boot ROM
POR (power on reset), Hardware and Watchdog reset **NAND boot**	1) ARSR[HWR] and [WDG] is read to ensure that POR/HWR reset has occurred. NOTE: This step only occurs for the PXA32x and PXA30x A1	
	2) Boot ROM checks boot state fuses to see if NAND or NOR boot 3) Full Boot process. ReadID issued from NFC to NAND device. OBM in Block 0 copied to iSRAM.	

Table 16: Overview of Resets and Power Modes (Continued)

Power Mode or Reset Exit	V2.xx Boot ROM	V3.xx Boot ROM
POR (power on reset), Hardware and Watchdog reset **NOR/XIP boot and OneNAND and mDOC**	1) Boot ROM checks boot state fuses to see if NAND or NOR boot 2) Full Boot process. Control is transferred to external XIP Flash on nCS2 address 0x1000_0000.	
GPIO Reset - **NAND boot**	1) ARSR[GPR] is read to ensure GPIO reset occurred. NOTE: This step occurs only for the PXA32x and PXA30x A1	
	2) Boot ROM checks boot state fuses to see if NAND or NOR boot 3) PSPR must contain the location of OBM (0x5C01_4000) **NOTE:** If PSPR does not contain OBM address the system does not boot	2) Boot ROM checks boot state fuses to see if NAND or NOR boot 3) Full Boot process. ReadID issued from NFC to NAND device. OBM in Block 0 copied to iSRAM.
GPIO Reset - **NOR/XIP boot and OneNAND and mDOC**	1) Boot ROM checks boot state fuses to see if NAND or NOR boot 2) Code resume to 0x1000_0000 nCS2 XIP NOR Flash	
S2/D3/C4 - **NAND boot**	1) ASCR[D3S] is read to ensure D3 mode exit has occurred. NOTE: This step occurs only for the PXA32x and PXA30x A1	
	2) PSPR must contain the location of OBM (0x5C01_4000) **NOTE:** If PSPR does not contain OBM address the system does not boot	2) AD3R[AD3_R0] register is checked to determine if the iSRAM retains state in S2/D3/C4 3) Boot ROM reads struct() at 0x5C00_8000
	NOTE: Refer to Section 5.8.2 for more details	
S2/D3/C4 - **NOR/XIP boot and OneNAND and mDOC**	1) Code resume to 0x1000_0000 nCS2 XIP NOR Flash	
After S2/D3/C4 exit ONLY; the MFPR[SLEEP_SEL] & ASCR[RDH] are cleared by Boot ROM		
S3/D4/C4 - **NAND boot**	1) ARSR[LPMR] is read to ensure D4 mode has occurred. NOTE: This step occurs only for the PXA32x and PXA30x A1	
	2) Boot ROM checks boot state fuses to see if NAND or NOR boot 3) Full Boot process. ReadID issued from NFC to NAND device. OBM in Block 0 copied to iSRAM.	

Table 16: Overview of Resets and Power Modes (Continued)

Power Mode or Reset Exit	V2.xx Boot ROM	V3.xx Boot ROM
S3/D4/C4 - **NOR/XIP boot and OneNAND and mDOC**	1) Boot ROM checks boot state fuses to see if NAND or NOR boot 2) Full Boot process. Control is transferred to external XIP Flash on nCS2 address 0x1000_0000	
NOTE: The GPIOs and register configurations as described in Section 5.9 are reloaded depending on the boot type for all the events listed in this table EXCEPT for S2/D3/C4 exit **NOTE:** OneNAND and mDOC power mode and reset exits differ between Boot ROM versions. Chapter 5, "Boot ROM Implementation Details," and Section 5 "Managed NAND Memory Support" .		

5.8.1 Platform Boot Process for Watchdog Reset, Power on Reset, Low Power Exit, and GPIO Reset

The power events for watchdog reset, power-on reset, and hardware reset are all handled the same way in the Boot ROM. A full boot is performed; however, a GPIO reset is a special case and the V2 and V3 Boot ROM address this reset differently.

See Table 9, Trusted Image Module Locations for Trusted Boot, on page 36 for details.

5.8.2 S2/D3/C4 Resume Requirements

The resume-from-sleep (S2/D3/C4) process is handled by the Boot ROM differently than other low-power events.

Several requirements need to be implemented to successfully resume a platform from S2/D3/C4 mode. Refer to the appropriate sections of this document, as well as the processor-specific development manuals to verify proper resume procedures. At a minimum:

- Verify any required step for S2/D3/C4 resume as described in the processor documentation.
- Review the settings of the ball functions in Section 5.9, "Boot ROM: Processor-Specific Configurations" and make any necessary changes before entering the sleep state to allow for a successful resume.
- Refer to the specific processor reference manuals to ensure that DDR memory, internal SRAM, and all clocks are handled correctly during S2/D3/C4 and resume.

Refer to the Clocks Controller and Power Management chapter in the *PXA3xx Family Processor Vol. I: System and Timer Configuration Developers Manual* for specific details for PXA3xx processor.

5.8.2.1 S2/D3/C4 with NAND Boot Version 2.xx Boot ROM

Version 2.xx NAND boot platforms resume only from S2/D3/C4 mode to ISRAM.

NAND platforms must prepare the platform for a successful resume process. Software must write the resume address to the PMU Scratch pad register (PSPR) before entering S2/D3/C4 mode. The address written to the PSPR must be the starting address of the OBM boot code in the ISRAM (0x5C01_4000). This code must successfully resume the platform from S2/D3/C4. Upon S2/D3/C4 exit, the Boot ROM reads the address stored in the PSPR and transfers control of the platform to this address.

The following requirements must be implemented in software to successfully resume from S2/D3/C4.

Prior to entry into S2/D3/C4:

1. Copy the OBM from block 0 of the NAND Flash to ISRAM at 0x5C01_4000
2. Store the address 0x5C01_4000 into the PSPR register
3. Configure the internal SRAM to retain state during S2 mode using AD3R[AD3_R0]
4. Enter S2/D3/D4 mode

Resuming from S2/D3/C4:

The OBM code must implement all necessary requirements for resuming the system as described in the processor specific development manuals.

5.8.2.2 S2/D3/C4 with NAND Boot Version 3.xx

Version 3.xx NAND boot platforms may resume to either ISRAM or load the OBM from NAND Block 0. Refer to Section 6, “Non-Trusted Image Module” and Section 7, “Trusted Image Module” for details on how to configure Resume packages.

5.8.2.3 S2/D3/C4 iSRAM resuming

The Boot ROM resumes from S2/D3/C4 mode and immediately reads iSRAM address 0x5C00_8000 for a structure. AD3R[AD3_R0] must be set to ensure state retention during S2 mode.

The structure is defined as follows:

```
struct{
Intversion;
Intidentifier;
Intstarting address;
Intsize;
}
```

Where:

`Version` – Hex version number with the format `0xaabb`

- `aa` – Major number
- `bb` – Minor number

For example, `0x112` for version 1.12

`Identifier` – String “`slpr`”

`Starting address` – Starting internal SRAM address of the resume code

`Size` – Size of the resume code

The following requirements must be implemented in software to successfully resume from S2/D3/C4.

Prior to entry into S2/D3/C4:

1. Copy the resume struct() to ISRAM at 0x5C00_8000
2. Configure the internal SRAM to retain state during S2 mode using AD3R[AD3_R0]
3. Enter S2/D3/D4 mode

Resuming from S2/D3/C4:

The OBM code must implement all necessary requirements for resuming the system as described in the processor-specific developers manual.

Note

If the Boot ROM fails to find the sleep-resume structure, then the OBM code from NAND Block 0 is copied back into iSRAM at address 0x5C01_3000.

Using the sleep struct() to manage the resume process eliminates the Boot ROM from reloading the OBM code from NAND Block 0 thus providing a shorter exit latency. As noted above, if the struct() is not found, then the resume period is extended by the time necessary to reload OBM.

5.9 Boot ROM: Processor-Specific Configurations

The following tables provide implementation settings for the PXA3xx Processor Family and Tavor processor.

Note

The Boot ROM configures the USB and UART ports as defined in this section. If another UART port is required, then the default port must be unconfigured. This unconfiguration can be managed using predefined packages in the NTIM/TIM header. Refer to Section 6.3, “Predefined Packages for Reserved Area”. Both ports (USB and FFUART) cannot be disabled.

PXA32x Boot ROM Register configurations:

- Table 17, “PXA32x Processor Implementation Settings"
- Table 18, “Chip Select 2 Setup"
- Table 19, “Additional PXA32x Processor Ball Values Set for NAND Platforms Only"
- Table 20, “FFUART Pins"
- Table 21, “USB Single Ended Pins"
- Table 22, “During Sleep (S3/D3/C4 mode) Resume"

PXA31x Boot ROM Register configurations:

- Table 23, “PXA31x Processor Implementation Settings"
- Table 24, “Additional PXA31x Processor Ball Values Set for NAND Platforms Only"
- Table 25, “USB Port ULPI Pins"
- Table 26, “FFUART Pins"

PXA30x Boot ROM Register configurations:

- Table 27, “PXA30x Processor Implementation Settings"
- Table 28, “Chip Select 2 Setup"
- Table 29, “Additional PXA30x Processor Ball Values Set for NAND Platforms Only"
- Table 30, “FFUART Pins"

Tavor Boot ROM Register configurations:

- Table 32, “Tavor Processor Register Settings"
- Table 33, “Additional Tavor Processor Ball Values Set for NAND Platforms Only"
- Table 34, “FFUART Pins (Primary Location)"
- Table 35, “FFUART Pins (Secondary Location)"
- Table 36, “USB 2.0 Pins"

5.9.1 PXA32x Processor Register Settings

Table 17: PXA32x Processor Implementation Settings

Ball Name	Address	Value
GPIO2	0x40E1_012C	0x0000_0001
GPIO3	0x40E1_0130	0x0000_0001
GPIO4	0x40E1_0134	0x0000_0001
nXCVREN	0x40E1_0138	0x0000_1900
nBE0	0x40E1_0214	0x0000_1800
nBE1	0x40E1_0218	0x0000_1800
nLUA	0x40E1_0234	0x0000_1900
nLLA	0x40E1_0238	0x0000_1900
DF_ADDR0	0x40E1_023C	0x0000_1800
DF_ADDR1	0x40E1_0240	0x0000_1800
DF_ADDR2	0x40E1_0244	0x0000_1800
DF_ADDR3	0x40E1_0248	0x0000_1800
DF_CLE_NOE	0x40E1_0204	0x0000_1800
DF_ALE_nWE1	0x40E1_0208	0x0000_1801
DF_IO0	0x40E1_024C	0x0000_1401
DF_IO1	0x40E1_0254	0x0000_1401
DF_IO2	0x40E1_025C	0x0000_1401
DF_IO3	0x40E1_0264	0x0000_1401
DF_IO4	0x40E1_026C	0x0000_1401
DF_IO5	0x40E1_0274	0x0000_1401
DF_IO6	0x40E1_027C	0x0000_1401
DF_IO7	0x40E1_0284	0x0000_1401
DF_IO8	0x40E1_0250	0x0000_1401
DF_IO9	0x40E1_0258	0x0000_1401
DF_IO10	0x40E1_0260	0x0000_1401
DF_IO11	0x40E1_0268	0x0000_1401
DF_IO12	0x40E1_0270	0x0000_1401
DF_IO13	0x40E1_0278	0x0000_1401
DF_IO14	0x40E1_0280	0x0000_1401
DF_IO15	0x40E1_0288	0x0000_1401

Table 18: Chip Select 2 Setup

Register	Address	Value	Function
SMEMC CSADRCFG2	0x4A00_0088	0x0032_0919	––
SMEMC MSC1	0x4A00_000C	Bits [15:0] = 0x7FF8 All others = Default	––

The ball values in Table 19 are in addition to the ball values in Table 17 and are only configured if the processor is configured for NAND boot support.

Table 19: Additional PXA32x Processor Ball Values Set for NAND Platforms Only

Ball Name	Address	Value
DF_ALE_nWE2	0x40E1_021C	[31:3] = Default; [2:0] = 1
DF_nCS0	0x40E1_0224	[31:3] = Default; [2:0] = 1
DF_nCS1	0x40E1_0228	[31:3] = Default; [2:0] = 1
DF_nWE	0x40E1_022C	[31:3] = Default; [2:0] = 1
DF_nRE	0x40E1_0230	[31:3] = Default; [2:0] = 1

Values in Table 20 are used for configuring the UART port for image download over UART.

Table 20: FFUART Pins

Ball Name	Address	Value	Function
GPIO41	0x40E1_0438	0x0000_1042	FFRXD
GPIO42	0x40E1_043C	0x0000_1042	FFTXD
GPIO43	0x40E1_0440	0x0000_1042	FFCTS
GPIO44	0x40E1_0444	0x0000_1042	FFDCD
GPIO45	0x40E1_0448	0x0000_1042	FFDSR
GPIO46	0x40E1_044C	0x0000_1042	FFRI
GPIO47	0x40E1_0450	0x0000_1042	FFDTR
GPIO48	0x40E1_0454	0x0000_1042	FFRTS

Values in Table 21 are used for configuring the USB Client single-ended port for image download over USB.

Table 21: USB Single Ended Pins

Ball Name	Address	Value	Function
GPIO97	0x40E1_054C	0x0000_1402	nOE2
GPIO98	0x40E1_0550	0x0000_1402	VPO2
GPIO99	0x40E1_0600	0x0000_1402	RCV2
GPIO100	0x40E1_0604	0x0000_1402	VMO2
GPIO102	0x40E1_060C	0x0000_1402	VM2
GPIO103	0x40E1_0610	0x0000_1402	VP2
GPIO104	0x40E1_0614	0x0000_1402	SPEED

5.9.2 Other Registers

Table 22: During Sleep (S3/D3/C4 mode) Resume

Ball Name	Address	Value	Function
D0CKEN_A	0x4134_000C	0xC078_0F10	—
D0CKEN_B	0x4134_0010	0x0000_00C0	—

5.10 PXA31x Processor Register Settings

Table 23 and Table 24 provide implementation settings for the PXA31x processor.

Table 23: PXA31x Processor Implementation Settings

Ball Name	Address	Value
GPIO0	0x40E1_00B4	0x0000_1401
GPIO1	0x40E1_00B8	0x0000_1501
GPIO2	0x40E1_00BC	0x0000_1501
nBE0	0x40E1_0204	0x0000_1400
nBE1	0x40E1_0208	0x0000_1400
nLUA	0x40E1_0244	0x0000_1500
nLLA	0x40E1_0254	0x0000_1500
DF_ADDR0	0x40E1_0210	0x0000_1400
DF_ADDR1	0x40E1_0214	0x0000_1400
DF_ADDR2	0x40E1_0218	0x0000_1400
DF_ADDR3	0x40E1_021C	0x0000_1400
DF_CLE_NOE	0x40E1_0240	0x0000_1500
DF_ALE_nWE	0x40E1_020C	0x0000_1501
DF_IO0	0x40E1_0220	0x0000_1801
DF_IO1	0x40E1_0228	0x0000_1801
DF_IO2	0x40E1_0230	0x0000_1801
DF_IO3	0x40E1_0238	0x0000_1801
DF_IO4	0x40E1_0258	0x0000_1801
DF_IO5	0x40E1_0260	0x0000_1801
DF_IO6	0x40E1_0268	0x0000_1801
DF_IO7	0x40E1_0270	0x0000_1801
DF_IO8	0x40E1_0224	0x0000_1801
DF_IO9	0x40E1_022C	0x0000_1801
DF_IO10	0x40E1_0234	0x0000_1801
DF_IO11	0x40E1_023C	0x0000_1801
DF_IO12	0x40E1_025C	0x0000_1801

Table 23: PXA31x Processor Implementation Settings (Continued)

Ball Name	Address	Value
DF_IO13	0x40E1_0264	0x0000_1801
DF_IO14	0x40E1_026C	0x0000_1801
DF_IO15	0x40E1_0274	0x0000_1801

The configurations in Table 24 are in addition to those in Table 24 and are only configured if the processor is configured for NAND boot support.

Table 24: Additional PXA31x Processor Ball Values Set for NAND Platforms Only

Ball Name	Address	Value
DF_nCS0	0x40E1_0248	[31:3] = Default; [2:0] = 1
DF_nCS1	0x40E1_0278	[31:3] = Default; [2:0] = 1
DF_nWE	0x40E1_00CC	[31:3] = Default; [2:0] = 1
DF_nRE	0x40E1_0200	[31:3] = Default; [2:0] = 1
Ball_DF_INT_RnB	0x40E1_00C8	0x0000_1500

The values in Table 25 are used for configuring the USB 2.0 Client ULPI port for image download over USB.

Table 25: USB Port ULPI Pins

Ball Name	Address	Value	Function
GPIO30	0x40E1_0418	0x0000_55C3	ULPI Data 0
GPIO31	0x40E1_041C	0x0000_55C3	ULPI Data 1
GPIO32	0x40E1_0420	0x0000_55C3	ULPI Data 2
GPIO33	0x40E1_0424	0x0000_55C3	ULPI Data 3
GPIO34	0x40E1_0428	0x0000_55C3	ULPI Data 4
GPIO35	0x40E1_042C	0x0000_55C3	ULPI Data 5
GPIO36	0x40E1_0430	0x0000_55C3	ULPI Data 6
GPIO37	0x40E1_0434	0x0000_55C3	ULPI Data 7
GPIO38	0x40E1_0438	0x0000_55C1	ULPI Clk

Values in Table 26 are used for configuring the UART port for image download over UART.

Table 26: FFUART Pins

Ball Name	Address	Value	Function
GPIO99	0x40E1_0600	[31:1] = Default; [0] = 0b1	FFRXD
GPIO100	0x40E1_0604	31:1] = Default; [0] = 0b1	FFTXD
GPIO101	0x40E1_0608	31:1] = Default; [0] = 0b1	FFCTS
GPIO106	0x40E1_061C	31:1] = Default; [0] = 0b1	FFRTS

5.11 PXA30x Processor Register Settings

Table 27 and Table 28 provide implementation settings for the PXA30x processor.

Table 27: PXA30x Processor Implementation Settings

Ball Name	Address	Value
GPIO0	0x40E1_00B4	0x0000_1401
GPIO1	0x40E1_00B8	0x0000_1501
GPIO2	0x40E1_00BC	0x0000_1501
nBE0	0x40E1_0204	0x0000_1400
nBE1	0x40E1_0208	0x0000_1400
nLUA	0x40E1_0244	0x0000_1500
nLLA	0x40E1_0254	0x0000_1500
DF_ADDR0	0x40E1_0210	0x0000_1400
DF_ADDR1	0x40E1_0214	0x0000_1400
DF_ADDR2	0x40E1_0218	0x0000_1400
DF_ADDR3	0x40E1_021C	0x0000_1400
DF_CLE_NOE	0x40E1_0240	0x0000_1500
DF_ALE_nWE	0x40E1_020C	0x0000_1501
DF_IO0	0x40E1_0220	0x0000_1801
DF_IO1	0x40E1_0228	0x0000_1801
DF_IO2	0x40E1_0230	0x0000_1801
DF_IO3	0x40E1_0238	0x0000_1801
DF_IO4	0x40E1_0258	0x0000_1801
DF_IO5	0x40E1_0260	0x0000_1801
DF_IO6	0x40E1_0268	0x0000_1801
DF_IO7	0x40E1_0270	0x0000_1801
DF_IO8	0x40E1_0224	0x0000_1801
DF_IO9	0x40E1_022C	0x0000_1801
DF_IO10	0x40E1_0234	0x0000_1801
DF_IO11	0x40E1_023C	0x0000_1801
DF_IO12	0x40E1_025C	0x0000_1801
DF_IO13	0x40E1_0264	0x0000_1801
DF_IO14	0x40E1_026C	0x0000_1801
DF_IO15	0x40E1_0274	0x0000_1801

Table 28: Chip Select 2 Setup

Register	Address	Value	Function
SMEMC_CSADRCFG_2	0x4A00_0088	0x00320919	—
SMEMC_MSC1	0x4A00_000C	Bits [15:0] = 0x7FF8 All others = Default	—

These configurations are in addition to the ones described in Table 29 and are only configured if the processor is configured for NAND boot support.

Table 29: Additional PXA30x Processor Ball Values Set for NAND Platforms Only

Ball Name	Address	Value
DF_nCS0	0x40E1_0248	[31:3] = Default; [2:0] = 1
DF_nCS1	0x40E1_0278	[31:3] = Default; [2:0] = 1
DF_nWE	0x40E1_00CC	[31:3] = Default; [2:0] = 1
DF_nRE	0x40E1_0200	[31:3] = Default; [2:0] = 1
DF_nCS0	0x40E1_0248	[31:3] = Default; [2:0] = 1
Ball_DF_INT_RnB	0x40E1_00C8	0x0000_1500

Values in Table 30 are used for configuring the UART port for image download over UART.

Table 30: FFUART Pins

Ball Name	Address	Value	Function
GPIO30	0x40E1_040C	0x0000_1402	FFRXD
GPIO31	0x40E1_0410	0x0000_1402	FFTXD
GPIO32	0x40E1_0414	0x0000_1502	FFCTS
GPIO33	0x40E1_0418	0x0000_1502	FFDCD
GPIO34	0x40E1_041C	0x0000_1502	FFDSR
GPIO35	0x40E1_0420	0x0000_1502	FFRI
GPIO36	0x40E1_0424	0x0000_1502	FFDTR
GPIO37	0x40E1_0428	0x0000_1502	FFRTS

Values in Table 31 are used for configuring the USB client single-ended port for image download over USB.

Table 31: USB Single Ended Pins

Ball Name	Address	Value	Function
GPIO99	0x40E1_0600	0x0000_1402	nOE2
GPIO100	0x40E1_0604	0x0000_1402	VPO2
GPIO101	0x40E1_0608	0x0000_1402	RCV2
GPIO102	0x40E1_060C	0x0000_1402	VMO2

Table 31: USB Single Ended Pins (Continued)

Ball Name	Address	Value	Function
GPIO104	0x40E1_0614	0x0000_1402	VM2
GPIO105	0x40E1_0618	0x0000_1402	VP2
GPIO106	0x40E1_061C	0x0000_1402	SPEED

5.12 Tavor Processor Implementation Details

The following tables provide implementation settings for the Tavor processor.

Table 32: Tavor Processor Register Settings

Ball Name	Address	Value
DF_nCS0	0x40E1_022C	0x0000_1001
DF_nWE	0x40E1_0234	0x0000_1001
DF_nRE_nOE	0x40E1_0238	0x0000_1001
DF_SCLK_E	0x40E1_0238	0x0000_1001
nLUA	0x40E1_0254	0x0000_1001
ND_CLE	0x40E1_020C	0x0000_1001
DF_nADV1_ALE	0x40E1_0218	0x0000_1001
DF_IO0	0x40E1_023C	0x0000_1001
DF_IO1	0x40E1_0240	0x0000_1001
DF_IO2	0x40E1_0244	0x0000_1001
DF_IO3	0x40E1_0248	0x0000_1001
DF_IO4	0x40E1_0264	0x0000_1001
DF_IO5	0x40E1_0268	0x0000_1001
DF_IO6	0x40E1_026C	0x0000_1001
DF_IO7	0x40E1_0270	0x0000_1001
DF_IO8	0x40E1_0274	0x0000_1041
DF_IO9	0x40E1_0278	0x0000_1041
DF_IO10	0x40E1_027C	0x0000_1041
DF_IO11	0x40E1_0280	0x0000_1041
DF_IO12	0x40E1_0284	0x0000_1041
DF_IO13	0x40E1_0288	0x0000_1041
DF_IO14	0x40E1_028C	0x0000_1041
DF_IO15	0x40E1_0290	0x0000_1041

These values are in addition to those inTable 32.

Table 33: Additional Tavor Processor Ball Values Set for NAND Platforms Only

Ball Name	Address	Value
DF_NADV2_ALE	0x40E1_0224	0x0000_1003
ND_INT_RnB	0x40E1_0228	0x0000_1C41
DF_ADDR2	0x40E1_025C	0x0000_1C40

Table 34: FFUART Pins (Primary Location)

Ball Name	Address	Value	Function
GPIO53	0x40E1_02D0	0x0000_0441	FFRXD
GPIO54	0x40E1_02CC	0x0000_0441	FFTXD
GPSRy	0x40E0_0018	0x0040_0000	
GPDRy	0x40E0_0010	0x0040_0000	

Table 35: FFUART Pins (Secondary Location)

Ball Name	Address	Value	Function
GPIO64	0x40E1_065C	0x0000_0442	FFRXD
GPIO63	0x40E1_0640	0x0000_0442	FFTXD
GPSRy	0x40E0_0018	0x8000_0000	
GPDRy	0x40E0_0010	0x8000_0000	

Table 36: USB 2.0 Pins

Ball Name	Address	Value	Function
Ball Name	Address	Value	Function
GPIO30	0x40E1_0618	0x0000_1C47	USB_ULPI_D1
GPIO31	0x40E1_0610	0x0000_1C44	USB_ULPI_D0
GPIO33	0x40E1_061C	0x0000_1C45	USB_ULPI_D2
GPIO34	0x40E1_0620	0x0000_1C55	USB_ULPI_D3
GPIO35	0x40E1_0628	0x0000_1C55	USB_ULPI_D4
GPIO36	0x40E1_062C	0x0000_1C55	USB_ULPI_D5
GPIO41	0x40E1_0614	0x0000_1C55	USB_ULPI_D6
GPIO38	0x40E1_0634	0x0000_1C54	USB_ULPI_CLK
GPIO39	0x40E1_0638	0x0000_1C54	USB_ULPI_STP
GPIO42	0x40E1_0624	0x0000_1C55	USB_ULPI_D7
GPIO37	0x40E1_0630	0x0000_1C54	USB_ULPI_DIR
GPIO40	0x40E1_063C	0x0000_1C54	USB_ULPI_NXT

5.13 Error Conditions

The Boot ROM reports error conditions by either writing the error code to address 0x5C00_8000 or via messaging using the WTPTP utility. Refer to Appendix A, Return Error Code Definitions.

5.14 Hints And Tips

Below is a short list of key information that must be understood and implemented in the system for a successful boot. Use these as a guide to start debugging a boot problem.

Some of these requirements may not be apparent when using the Marvell BSP. For example, the BSP would automatically configure the system to ensure a successful boot such as pre-pending the OBM image.

If using the processor without an OS BSP provided by Marvell or bringing up a proprietary OS, system developers must be aware of these high level but key points.

Version 2.xx processors:

- Require two words of offset to be pre-pended by the OBM image
- The OBM may occupy up to 2 blocks of NAND memory

Version 3.xx processors:

- Removed requirement for first 2 words

Both versions; Unused pages within the first block MUST be programmed (0b0) by padding the OBM image to equal 1 block size

- Required to make the ECC information consistent
- If the "unused" pages in Block 0 are not padded, the Boot ROM returns a failure and the system does not boot

The Boot ROM returns an error code in the event of a failed operation such as failure to boot, download an image or a memory read/write operation

- These error codes are written to ISRAM location 0x5C00_8000
- Error messages are also sent using the WTPTP utility
- Error Codes are listed in Appendix A of this document

6 Non-Trusted Image Module

The non-trusted boot solution uses the non-trusted image module (NTIM) format, which is shown in Section 6.1. This structure is dynamic because it is a packed structure of variable size, based on the information it contains. The maximum size of the structure for version 3.x of the Boot ROM is 4 KB.

The non-trusted image structure can be created by using the Marvell® Wireless Trusted Platform Tool Package or a custom tool created by the Original Equipment Manufacturer (OEM). The OEM can use the reserved area for value-added features; see Section 6.2, Reserved Area. The fields of the non-trusted image module structure define version, flash, and image information as shown below in Section 6.1 and described in:

- Section 6.1.1, Version Information
- Section 6.1.2, Flash Information
- Section 6.1.3, NTIM Sizing Information
- Section 6.1.4, Image Information Array

6.1 Non-Trusted Image Module Format

The non-trusted image module format for non-trusted boot operations is described below. All values are 32-bit unsigned integers.

```
VERSION INFORMATION:
unsigned int Version;
unsigned int Identifier;
unsigned int Trusted;
unsigned int IssueDate;
unsigned int OEMUniqueID;
FLASH INFORMATION:
unsigned int Reserved;
unsigned int Reserved;
unsigned int Reserved;
unsigned int Reserved;
unsigned int Reserved;
unsigned int BootFlashSign;
NTIM SIZING INFORMATION:
unsigned int NumImages;
unsigned int Reserved;
unsigned int SizeOfReserved;
IMAGE INFORMATION ARRAY IMAGE[NumImages];
unsigned int ImageID;
unsigned int NextImageID;
unsigned int FlashEntryAddr;
unsigned int LoadAddr;
unsigned int ImageSize;
unsigned int Reserved; // must be set to 0x0
unsigned int Reserved[9]
unsigned int Reserved[SizeOfReserved];
```

6.1.1 Version Information

The Version Information defines the following variables that provide version information about the non-trusted image module and platform:

- `Version` – Current version of the non-trusted image module.
- `Identifier` – `TIMH` identifier used to identify the structure.
- `Trusted` – Identifier that defines a trusted or a non-trusted platform.
- `IssueDate` – Date on which this module was created.
- `OEMUniqueID` – OEM-specific identifier.

6.1.2 Flash Information

The Flash Information defines the following variables and data structures that identify certain properties of the boot flash and the location of the Marvell Wireless Trusted Module (Marvell WTM) save state:

- `Reserved[5]` – Data structure that reserves five words to maintain compatibility with the trusted image module.
- `BootFlashSign` – A signature that indicates the flash from which the platform boots. The upper three bytes contain ASCII-encoded hexidecimal values of `XIP` for XIP or `NAN` for NAND, Samsung OneNAND™, and SanDisk® (formerly MSystems®) flash. The lower byte is the platform fuse encoding that determines the flash device being used. The BootFlashSign values are noted in Table 37

Table 37: BootFlashSign Definitions

BootFlashSign value	Platform Boot Device	Encoded HEX Value
0x4E41_4E06	x8 NAND device on DF_nCS0	NAN'06
0x4E41_4E04	x16 NAND device on DF_nCS0	NAN'04
0x4E41_4E01	x16 SanDisk mDOC on SMEMC nCS2	NAN'01
0x4E41_4E02	x16 OneNAND on SMEMC on nCS2	NAN'02
0x5849_5005	x16 XIP NOR XIPB Flash on nCS2	XIP'05
0x5849_5007	x16 XIP NOR XIPA Flash on nCS2	XIP'07

NOTE: XIPB and XIPA support different command sets. Refer to Table 14 for further information. Other NOR flash memories may be used.

NOTE: Refer to Table 14, Flash Commands Supported by the Boot ROM, on page 42 for descriptions of XIPA and XIPB flash memory

6.1.3 NTIM Sizing Information

- `NumImages` – Number of "`IMAGE INFORMATION`" substructures in the trusted image module.
- `Reserved` – Reserved.
- `SizeOfReserved` – Size of the reserved area; values can range from `0` to `4` KB - (size of other trusted image module information).

6.1.4 Image Information Array

The Image Information array contains information about each image loaded into the boot flash. The number of substructures is determined by the `NumImages` field described in the previous section.

- `ImageID` – Unique identifier of the image. The `"OBMI"` identifier must be present in the array for the Boot ROM to correctly boot the platform. Other identifiers can be determined by the OEM, but are limited to 32 bits in size.
- `NextImageID` – Next image that should be loaded from flash.
- `FlashEntryAddr` – Offset from the start of the boot flash pointed to by the `BootFlashSign` field.
- `LoadAddr` – Load address for the image; this can be DDR memory or internal SRAM.
- `ImageSize` – Size of the image in bytes.
- `Reserved` – Must be set to 0x0.

- `Reserved[9]` – Nine reserved words of data.

When creating the non-trusted image module, no padding should be used. This is a packed structure that should contain only the necessary information. For a successful operation with the Boot ROM, the structure must contain complete version and flash information, and at least one image information structure for the OEM boot module.

6.1.5 Reserved[SizeOfReserved]

- `Reserved[SizeOfReserved]` – Array of integers to be used by the OEM for value-added features.

6.2 Reserved Area

The Reserved Area is a dedicated space in the non-trusted image module that allows an OEM to add data that is targeted for specific use without altering the predefined layout of the non-trusted image module. The reserved area may be of variable size, which is tabulated in the `SizeOfReserved` field.

The content of the Reserved Area may be formatted as the OEM chooses; however, to be compatible with the Wireless Trusted Platform Tool Package set of tools, a predefined format has been established. This format consists of the Reserved Area Header and the Reserved Area Packages, described below.

6.2.1 Reserved Area Header

The Reserved Area Header spans eight bytes; its primary purpose is to indicate to the interpreter of the non-trusted image module that this portion of the reserved area complies with the format defined by the
Wireless Trusted Platform Tool Package-defined format. It also indicates the number of packages to follow. The structure for the Reserved Area Header is:

```
WTP_RESERVED_AREA:
    unsigned int WTPTP_Reserved_Area_ID;
    unsigned int NumReservedPackages;
```

- `WTPTP_Reserved_Area_ID` – This indicates to the interpreting software that the reserved area complies with the format defined by the Wireless Trusted Platform Tool Package. This value should be `0x4F505448`, which represents `OPTH` in ASCII.
- `NumReservedPackages` – The number of packages to follow.

6.2.2 Reserved Area Packages

The Reserved Area Packages are the building blocks of the reserved area. Each package consists of a header that identifies the content, size, and payload data.

```
WTP_RESERVED_AREA_HEADER:
       unsigned int Identifier;
       unsigned int Size;
```

- `Identifier` – The identifier that defines the type of the package.
- `Size` – The total size of the package: four bytes for the identifier, four bytes for the size, and the number of bytes of information in the payload that follows.

There may be an unlimited number of Reserved Area Packages as long as the size of the non-trusted image module does not exceed 4 KB.

6.3 Predefined Packages for Reserved Area

A number of packages have already been defined for use with the Wireless Trusted Platform Tool Package tools as shown in Table . These predefined packages and their associated predefined header identifiers are described below. The Boot ROM executes these packages whenever the Boot ROM controls system resources, this includes all modes except for S2/D3/C4 mode.

There are further optional packages described in 13.1 "Optional Settings in the TIM/NTIM Modules" .

Refer to the Wireless Trusted Platform documentation for more detail on packages.

6.3.1 GPIO Packages

The GPIO Packages allow users to set any memory space or address space to a preferred value. This means that GPIOs can be configured by the Boot ROM.The header ID for this package is 0x4750494F, which represents GPIO in ASCII. The number of pairs to set is defined by `NumGpios`, as shown in the code below.

```
OPT_GPIO_SET:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int NumGpios;

GPIO_DEF:
    volatile int *Addr;
    unsigned int Value;
```

This means that GPIOs can be configured by the Boot ROM. For example, a GPIO must be set high in a platform design using the PXA31x processor. The appropriate GPIO is selected - GPIO<76> - in this example. The GPIO package layout is shown below. This is added to the end of the NTIM header after the image details.

Reserved Data:

```
0x4F505448 // Package structure identifier

0x00000002 // number of packages, note must include termination package

0x4750494f // GPIO package

0x00000024// size of GPIO package (a package has address and data, known as a pair.
Each Pair is 8 bytes - 4 bytes address and 4 bytes data)

0x00000003// number of pairs (in this example there are 3 data pairs below)

0x40e104d0// register address (MFPR for GPIO<76>)

0x00000840// data

0x40e00014// register address (GPDR2)

0x00001000// data (set GPIO<76> as output)

0x40e00020// register address (GPSR)

0x00001000// data (set GPIO<76> as high)

0x5465726D // Termination package

0x00000008
```

6.3.2 UART/USB Protocol Packages

The UART/USB Protocol packages allow users to override the default USB and UART connection settings. The identifier for this package can be 0x55415254 (for UART) or 0x00555342 (for USB). The port may also select as appropriate one of the following IDs:

FFIDENTIFIER:	0x00004646
ALTIDENTIFIER:	0x00414C54
DIFFIDENTIFIER:	0x44696666
SEIDENTIFIER:	0x00005345
U2DIDENTIFIER:	0x55534232
CI2IDENTIFIER	0x00434932

```
OPT_PROTOCOL_SET:
WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int Port;
    unsigned int Enabled;
```

6.3.3 DDR Package

The optional DDR package queues the Boot ROM to set up DDR based on the supplied timing parameters. Only the timing parameters based on the targeted hardware apply. The ID for this package is 0x44447248, which represents DDRH in ASCII.

```
OPT_DDR_SET:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int ACCR_VALUE;
    unsigned int MDCNFG_VALUE;
    unsigned int DDR_HCAL_VALUE;
    unsigned int MDREFR_VALUE;
```

6.3.4 Resume Package

The Resume Package provides instructions to the Boot ROM for loading and resuming an image after a wake-up from sleep. The image address, size, and checksum are specified in this package. The ID for this package is 0x5265736D, which represents RESM in ASCII.

```
OPT_RESM_LOC:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int ImageAddr;
    unsigned int ImageSize;
    unsigned int ImageCRC;
```

6.3.5 USB Vendor Request Package

The USB Vendor Request package is included in the reserved data when a special package requested by the vendor is required. This structure is the first word of the any trailing data. There is no restriction that the data has to be 32-bit aligned.

The ID for this package is 0X56524551, which represents VREQ in ASCII.

```
USB_VENDOR_REQ:
WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int bmRequestType;
    unsigned int bRequest;
    unsigned int wValue;
```

```
unsigned int wIndex;
unsigned int wLength;
unsigned int wData;
```

6.4 Summary of Predefined Package IDs for the Non-Trusted Image Module

The following list summarizes the predefined package IDs for the non-trusted image module, as indicated in the header of each package.

Table 38: Reserved Area Predefined Package ID's

Name	Hex Word Value
DDRID	0x44447248
TERMINATORID	0x5465726D
GPIOID	0x4750494F
UARTID	0x55415254
USBID	0x00555342
RESUMEID	0x5265736D
USBVENDORREQ	0x56524551
USB_DEVICE_DESCRIPTOR	0x55534200
USB_CONFIG_DESCRIPTOR	0x55534201
USB_INTERFACE_DESCRIPTOR	0x55534202
USB_LANGUAGE_STRING_DESCRIPTOR	0x55534203
USB_MANUFACTURER_STRING_DESCRIPTOR	0x55534204
USB_PRODUCT_STRING_DESCRIPTOR	0x55534205
USB_SERIAL_STRING_DESCRIPTOR	0x55534206
USB_INTERFACE_STRING_DESCRIPTOR	0x55534207
USB_DEFAULT_STRING_DESCRIPTOR	0x55534208
USB_ENDPOINT_DESCRIPTOR	0x55534209

7 Trusted Image Module

The trusted boot solution uses the trusted image module format, which is shown in Section 7.1. This structure is dynamic because it is a packed structure of variable size, based on the information it contains. The maximum size of the structure for version 3.x of the Marvell® Trusted Boot ROM is 4 KB.

The trusted image structure can be created by using the Wireless Trusted Platform Tool Package or a custom tool created by the Original Equipment Manufacturer (OEM). The OEM can use the reserved area for value-added features; see Section 7.2, Reserved Area. The fields of the trusted image structure define version, flash, TIM sizing, and image information as shown in Section 7.1.1, Version Information, on page 72, through Section 7.1.7, Platform Digital Signature Information, on page 74.

7.1 Trusted Image Module Format

For the trusted image module format for the boot operation, all values are 32-bit unsigned integers.

VERSION INFORMATION:
unsigned int Version;
unsigned int Identifier;
unsigned int Trusted;
unsigned int IssueDate;
unsigned int OEMUniqueID;
FLASH INFORMATION:
unsigned int WTMFlashSign;
unsigned int WTMEntryAddr;
unsigned int WTMEntryAddrBack;
unsigned int Reserved[2];
unsigned int BootFlashSign;
TIM SIZING INFORMATION:
unsigned int NumImages;
unsigned int NumKeys;
unsigned int SizeOfReserved;
IMAGE INFORMATION ARRAY IMAGE[NumImages];
unsigned int ImageID;
unsigned int NextImage;
unsigned int FlashEntryAddr;
unsigned int LoadAddr;
unsigned int ImageSize;
unsigned int ImageSizeToHash;
unsigned int HashAlgorithmID;
unsigned int Hash[8];
KEY INFORMATION ARRAY KEY[NumKeys];
unsigned int KeyID;
unsigned int HashAlgorithmID;
unsigned int ModulusSize;
unsigned int PublicKeySize;
unsigned int RSAPublicExponent[64];
unsigned int RSAModulus[64];
unsigned int KeyHash[8];
unsigned int Reserved[SizeOfReserved];
PLATFORM DIGITAL SIGNATURE INFORMATION:
unsigned int DSAlgorithmID;
unsigned int HashAlgorithmID;
unsigned int ModulusSize;
unsigned int Hash[8];
unsigned int RSAPublicExponent[64];
unsigned int RSAModulus[64];
unsigned int RSADigS[64];

7.1.1 Version Information

The Version Information field provides information about the trusted image module and platform.

- `Version` – Current version of the trusted image module.
- `Identifier` – `TIMH` identifier used to locate the trusted image module.
- `Trusted` – Identifier to distinguish between trusted and non-trusted platforms.

- `IssueDate` – Date this module was created.
- `OEMUniqueID` – OEM-specific identifier.

7.1.2 Flash Information

The Flash Information substructure identifies boot flash properties and the location of the Wireless Trusted Module (Marvell WTM) save state, as follows.

- `WTMFlashSign` – Signature that provides the Marvell WTM save state location. The upper three bytes contain an ASCII-encoded hexadecimal value of `XIP` for XIP or `NAN` for NAND, Samsung OneNAND™, and SanDisk® (formerly MSystems) flash. The lower byte is the platform fuse encoding that determines the flash device being used.
- `WTMEntryAddr` – Address offset from the base of the flash device to the location of the primary Marvell WTM save state block.
- `WTMEntryAddrBack` – Address offset from the base of the flash device to the location of the backup Marvell WTM save state block.
- `Reserved[2]` – Reserved for future use.
- `BootFlashSign` – Signature that determines from which flash the platform boots. The upper three bytes contains an ASCII-encoded hexadecimal value of `XIP` for XIP or `NAN` for NAND, OneNAND, and SanDisk flash. The lower byte contains the platform fuse encoding that determines the flash device being used. The BootFlashSign values are noted in Table 39.

Table 39: BootFlashSign Definitions

BootFlashSign value	Platform Boot Device	Encoded HEX Value
0x4E41_4E06	x8 NAND device on DF_nCS0	NAN'06
0x4E41_4E04	x16 NAND device on DF_nCS0	NAN'04
0x4E41_4E01	x16 SanDisk mDOC on SMEMC nCS2	NAN'01
0x4E41_4E02	x16 OneNAND on SMEMC on nCS2	NAN'02
0x5849_5005	x16 XIP NOR XIPB Flash on nCS2	XIP'05
0x5849_5007	x16 XIP NOR XIPA Flash on nCS2	XIP'07
NOTE: XIPB and XIPA support different command sets. Refer to Table 14 for further information. Other NOR flash memories may be used.		

7.1.3 TIM Sizing Information

- `NumImages` – Number of "`IMAGE INFORMATION`" substructures in the trusted image module.
- `NumKeys` – Number of "`KEY INFORMATION`" substructures in the trusted image module.
- `SizeOfReserved` – Size of the reserved area; values can range from 0 to 4 KB – (size of other trusted image module information).

7.1.4 Image Information Array

The Image Information Array is a substructure that contains information about each image loaded into the boot flash. The number of substructures is determined by the `NumImages` field above.

- `ImageID` – A unique identifier for the image. The `"OBMI"` identifier must be present in the array for the Boot ROM to correctly boot the platform. Other identifiers can be determined by the OEM, but are limited to 32 bits in size.
- `NextImageID` – Next image that should be loaded from flash memory.

- FlashEntryAddr – Offset from the start of the boot flash pointed to by the BootFlashSign field.
- LoadAddr – Absolute address for the image, which can be a DDR memory, internal SRAM, or XIP flash address.
- ImageSize – Size of the image in bytes.
- ImageSizeToHash – Number of bytes of the image that are included in the hash below.
- HashAlgorithmID – Hashing algorithm that is used; values are 160 for SHA-1 and 256 for SHA-2.
- Hash[8] – Array that holds the hash of the image.

7.1.5 Key Information Array

The Key Information Array is an array of "KEY INFORMATION" substructures. The number of substructures is determined by the NumKeys field.

- KeyID – Identifier for this key.
- HashAlgorithmID – Hashing algorithm that is used; values are 160 for SHA-1 and 256 for SHA-2.
- ModulusSize – Size of the RSA modulus; maximum of 2048 bits.
- PublicKeySize – Size of the RSA public key; maximum of 2048 bits.
- RSAPublicExponent[64] – RSA public exponent.
- RSAModulus[64] – RSA modulus.
- KeyHash[8] – SHA-1 or SHA-2 hash value.

7.1.6 Reserved[SizeOfReserved]

- Reserved[SizeOfReserved] – Array of integers to be used by the OEM for value-added features.

7.1.7 Platform Digital Signature Information

The Platform Digital Signature Information substructure holds the security information for this trusted image module.

- DSAlgorithmID – Digital Signature algorithm identifier.
- HashAlgorithmID – Hashing algorithm identifier; values are 160 for SHA-1 and 256 for SHA-2.
- ModulusSize – Size of the RSA modulus.
- Hash[8] – Hash value of the digital signature key.
- RSAPublicExponent[64] – Public exponent.
- RSAModulus[64] – RSA modulus.
- RSADigS[64] – Encrypted digital signature.

When creating the trusted image module, no padding is used. This is a packed structure that should contain only the necessary information. For a successful operation with the Boot ROM, the minimum requirements are:

- Complete version information.
- Complete flash memory information.
- At least one image information structure for the OEM boot module.
- At least one key information structure for the JTAG reenablement key.
- Correct RSA digital signature using the PKCS 1 method.

7.2 Reserved Area

The Reserved Area is a dedicated space in the trusted image module (TIM) that allows an OEM to add data that is targeted for specific use without altering the predefined layout of the trusted image module. The reserved area may be of variable size, which is tabulated in the `SizeOfReserved` field under the `"FlashInformation"` structure.

The content of the Reserved Area may be formatted as the OEM chooses, but to be compatible with the Wireless Trusted Platform Tool Package set of tools, a predefined format has been established. This predefined format consists of the Reserved Area Header and the Reserved Area Packages, described below.

7.2.1 Reserved Area Header

The Reserved Area Header component spans eight bytes; its primary purpose is to indicate to the interpreter of the non-trusted image module that this portion of the reserved area complies with the format defined by the Wireless Trusted Platform Tool Package. It also indicates the number of packages to follow. The structure for the Reserved Area Header is as follows:

```
WTP_RESERVED_AREA:
    unsigned int WTPTP_Reserved_Area_ID;
    unsigned int NumReservedPackages;
```

- `WTPTP_Reserved_Area_ID` – This indicates to the interpreting software that the reserved area complies with the format defined by the Wireless Trusted Platform Tool Package. This value should be `0x4F505448`, which represents `OPTH` in ASCII.
- `NumReservedPackages` – The number of packages to follow.

7.2.2 Reserved Area Packages

The Reserved Area Packages are the building blocks of the reserved area. Each package consists of a header to identify the content, size, and payload data.

```
WTP_RESERVED_AREA_HEADER:
    unsigned int Identifier;
    unsigned int Size;
```

- `Identifier` – The identifier that defines the type of the package.
- `Size` – The total size of the package: four bytes for the identifier, four bytes for the size, and the number of bytes of information in the payload that follows.

There may be an unlimited number of Reserved Area Packages as long as the size of the non-trusted image module does not exceed 4 KB.

7.3 Predefined Packages

A number of packages have already been defined for use with the Wireless Trusted Platform Tool Package tools as shown on the following page. These predefined packages and their associated predefined header identifiers are described below.

There are additional optional packages described in 13.1 "Optional Settings in the TIM/NTIM Modules" .

Refer to the Wireless Trusted Platform documentation for more detail on packages.

7.3.1 GPIO Packages

The GPIO packages allow users to set any memory space or address space to a preferred value, thereby allowing GPIOs to be configured by the Boot ROM.The header ID for this package is 0x4750494F, which represents GPIO in ASCII. The number of pairs to set is defined by NumGpios, as shown in the code below.

```
OPT_GPIO_SET:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int NumGpios;

GPIO_DEF:
    volatile int *Addr;
    unsigned int Value;
```

GPIOs can now be configured by the Boot ROM. For example, a GPIO is required to be set high in a platform design using the PXA31x processor. The appropriate GPIO is selected - GPIO<76>, in this example. The GPIO package layout is shown below. This information is added to the end of the NTIM header after the image details.

Reserved Data:

```
0x4F505448 // Package structure identifier

0x00000002 // number of packages, note must include termination package

0x4750494f // GPIO package

0x00000024// size of GPIO package (a package has address and data, known as a pair.
Each Pair is 8 bytes - 4 bytes address and 4 bytes data)

0x00000003// number of pairs (in this example there are 3 data pairs below)

0x40e104d0// register address (MFPR for GPIO<76>)

0x00000840// data

0x40e00014// register address (GPDR2)

0x00001000// data (set GPIO<76> as output)

0x40e00020// register address (GPSR)

0x00001000// data (set GPIO<76> as high)

0x5465726D // Termination package

0x00000008
```

7.3.2 UART/USB Protocol Packages

The UART/USB Protocol Packages allow overriding default USB and UART connection settings. The identifier for this package is `0x55415254` (for UART) or `0x00555342` (for USB). The port may also select as appropriate one of the following IDs:

FFIDENTIFIER:	0x00004646
ALTIDENTIFIER:	0x00414C54
DIFFIDENTIFIER:	0x44696666

SEIDENTIFIER:	0x00005345
U2DIDENTIFIER:	0x55534232
CI2IDENTIFIER	0x00434932

```
OPT_PROTOCOL_SET:
WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int Port;
    unsigned int Enabled;
```

7.3.3 DDR Package

The optional DDR package queues the Boot ROM to set up DDR based on the supplied timing parameters. Only the timing parameters based on the targeted hardware apply. The ID for this package is 0x44447248 which represents DDRH in ASCII.

```
OPT_DDR_SET:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int ACCR_VALUE;
    unsigned int MDCNFG_VALUE;
    unsigned int DDR_HCAL_VALUE;
    unsigned int MDREFR_VALUE;
```

7.3.4 Resume Package

The Resume package provides instructions to the Boot ROM for loading and resuming an image after a wake-up from sleep. The image address, size, and checksum are specified in this package. The ID for this package is 0x5265736D, which represents RESM in ASCII.

```
OPT_RESM_LOC:
    WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int ImageAddr;
    unsigned int ImageSize;
    unsigned int ImageCRC;
```

7.3.5 Autobind Package

The Autobind package has to be included in the reserved area for the Boot ROM to allow the OEM Boot Module to run on the XScale® core during provisioning. The ID for this package is 0X42494e44, which represents BIND in ASCII.

```
OPT_AUTOBIND:
WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int AutoBind;
```

7.3.6 USB Vendor Request Package

The USB Vendor Request package is included in the reserved data when a special package requested by the vendor is required. This structure is the first word of the any trailing data. There is no restriction that the data has to be 32-bit aligned.

The ID for this package is 0X56524551, which represents VREQ in ASCII.

```
USB_VENDOR_REQ:
WTP_RESERVED_AREA_HEADER WRAH;
    unsigned int bmRequestType;
    unsigned int bRequest;
    unsigned int wValue;
    unsigned int wIndex;
```

```
unsigned int wLength;
unsigned int wData;
```

7.4 Hashing Methods

The following provides the fields used to create each hash located in the trusted image module.

- Hash Value of the Trusted Image Module – The hash value of the trusted image module covers all information up to the platform digital signature substructure. The size is determined dynamically as follows:

  ```
  Trusted image module size to hash = size of version information + size of flash information +
  (size of image information * NumImages) +
  (size of key information * NumKeys)+ size of reserved + 3 Hash_value = SHA-1 (start of trusted image module through trusted image module size to hash)
  ```

- Hash Value of the Image – The hash value of the image is taken from the start of the binary over the size indicated by the field `ImageSizeToHash`.

  ```
  Hash value = SHA-1(start of image thru (ImageSize - ImageSizeToHash))
  ```

- Hash Value of the Keys – The hash value of the keys is taken over the `RSAPublicExponent` array and the RSA modulus array. There are two hashing methods used, depending on the key being hashed. This is a requirement based on the trusted platform module implementation.
  - JTAG Re-enabling, OEM Platform Bind, and Trusted Certificate Authority keys
  - `Hash value = SHA1 (RSAPublicExponent[64]/RSAModulus[64]/SHA padding)`

7.5 Summary of Predefined Package IDs for the Trusted Image Module

The following list summarizes the predefined package IDs as indicated in the header of each package.

Table 40: Reserved Area Predefined Package ID's

Name	Hex Word Value
DDRID	0x44447248
AUTOBIND	0X42494e44
TERMINATORID	0x5465726D
GPIOID	0x4750494F
UARTID	0x55415254
USBID	0x00555342
RESUMEID	0x5265736D
USBVENDORREQ	0x56524551
USB_DEVICE_DESCRIPTOR	0x55534200
USB_CONFIG_DESCRIPTOR	0x55534201
USB_INTERFACE_DESCRIPTOR	0x55534202
USB_LANGUAGE_STRING_DESCRIPTOR	0x55534203
USB_MANUFACTURER_STRING_DESCRIPTOR	0x55534204
USB_PRODUCT_STRING_DESCRIPTOR	0x55534205

Table 40: Reserved Area Predefined Package ID's (Continued)

Name	Hex Word Value
DDRID	0x44447248
USB_SERIAL_STRING_DESCRIPTOR	0x55534206
USB_INTERFACE_STRING_DESCRIPTOR	0x55534207
USB_DEFAULT_STRING_DESCRIPTOR	0x55534208
USB_ENDPOINT_DESCRIPTOR	0x55534209

8 Non-Trusted Operation

This section provides usage models for non-trusted boot operation.

8.1 Operation with a Non-Trusted Image Module

Operation with a non-trusted image module (NTIM) provides flexibility in a platform design. The image module allows an OEM to reuse the same binary images on both non-trusted and trusted platforms, which greatly reduces the engineering effort required to support a family of products.

The non-trusted image module contains image information that provides some level of checks before the image is booted. An image size and Cyclic Redundancy Check (CRC) enable some level of error detection at boot time. Information about the image size and location enables the OEM boot module to grow and change as development continues.

If a non-trusted image module is not used, certain restrictions are imposed on the image size by the Marvell® Trusted Boot ROM. When developing a platform, these restrictions should be considered carefully due to differences in compatibility between trusted and non-trusted image modules.

8.1.1 NAND Flash

The Trusted Boot ROM supports booting from x8 or x16 NAND devices attached to Chip Select 0 of the data flash controller of the processor. Both large- and small-block devices are supported; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image module is expected to be located in Block 0 at offset `0x0` of the NAND device. The Trusted Boot ROM loads the first page of Block 0 and searches for the `"TIMH"` identifier embedded in the version information of the non-trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the non-trusted image module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array; it is identified by the `"OBMI"` image identifier. This is the only required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

After the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the `ImageSize`, `LoadAddr`, and `ImageSizeToCRC` entries. The calculated CRC is then compared to the CRC stored in the non-trusted image module, in the `CRC` field. Upon a successful check, control is transferred to the image at the load address.

8.1.2 XIP Flash on Chip Select 2

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 2 of the processor static memory controller. There is support for several XIP devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image module is expected to be located at offset `0x0` of the XIP device. For Chip Select 2, the XIP device is memory-mapped to `0x1000_0000`. This is the address where the non-trusted image module must reside. The Trusted Boot ROM searches for the `"TIMH"` identifier embedded in the version information of the non-trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the non-trusted image module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array. It is identified by the "OBMI" image identifier. This is the only required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by FlashEntryAddr to the location pointed to by LoadAddr. The number of bytes loaded is determined by the ImageSize entry.

After the image has been loaded to the correct address, the CRC's ImageSizeToCRC must be verified. The image CRC is calculated based on the ImageSize and LoadAddr entries. The calculated CRC is then compared to the CRC stored in the non-trusted image module, in the CRC field. Upon a successful check, control is transferred to the image at the load address.

8.1.3 XIP Flash on Chip Select 0

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 0 of the processor static memory controller. There is support for several XIP devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image module is expected to be located at offset 0xC000 of the XIP device. For Chip Select 0, the XIP device is memory-mapped to 0x0000_0000, so the non-trusted image module must reside in 0x0000_C000. The Trusted Boot ROM searches for the "TIMH" identifier embedded in the version information of the non-trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the non-trusted image module to load the OEM boot module.

The OEM boot module is described by the image information contained in the "IMAGE INFORMATION ARRAY". It is identified by the "OBMI" image identifier. This is the only required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by FlashEntryAddr to the location pointed to by LoadAddr. The number of bytes loaded is determined by the ImageSize entry.

After the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the ImageSize and LoadAddr entries. The calculated CRC is then compared to the CRC stored in the non-trusted image module, in the CRC field. Upon a successful check, control is transferred to the image at the load address.

8.1.4 Samsung OneNAND* Flash

The Trusted Boot ROM supports booting from a x16 OneNAND device attached to Chip Select 2 of the processor static memory controller. There is support for large-block devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image module is expected to be located in Block 0 at offset 0x0 of the OneNAND device. The OneNAND device is memory-mapped to 0x1000_0000. The Trusted Boot ROM loads the first page of Block 0 and searches for the "TIMH" identifier embedded in the version information of the non-trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the non-trusted image module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array, and is identified by the "OBMI" image identifier. This is the only required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by FlashEntryAddr to the location pointed to by LoadAddr. The number of loaded bytes is determined by the ImageSize entry.

After the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the ImageSize, LoadAddr, and ImageSizeToCRC entries. The calculated CRC is then compared to the CRC stored in the non-trusted image module, in the CRC field. Upon a successful check, control is transferred to the image at the load address.

Table 5.5, OneNAND Support, on page 43 specifies the Device ID's and accompanying stepping generation codes that are supported.

8.1.5 SanDisk* Flash

The Trusted Boot ROM supports booting from the x16 SanDisk device attached to Chip Select 2 of the processor static memory controller. There is support for large block devices. Contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image module is expected to be located in Partition 2, Sector 0 of the SanDisk mDOC device. The MSystems device is memory-mapped to `0x1000_0000`. The Trusted Boot ROM loads the first page of Block 0 and searches for the `"TIMH"` identifier embedded in the version information of the non-trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the non-trusted image module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array, and is identified by the `"OBMI"` image identifier. This is the only required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

After the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the `ImageSize`, `LoadAddr`, and `ImageSizeToCRC` entries. Then, the calculated CRC is compared to the CRC stored in the non-trusted image module, in the `CRC` field.

8.1.6 Image Downloading

The Trusted Boot ROM enables the differential USB 1.1 client and the full-featured universal asynchronous receiver-transmitter (FFUART) port shortly after a power-on reset. An image can be downloaded over one of these ports by using the `WTPTP.exe` utility as described in 12 "Host Tools" .

The use of the non-trusted image module is required for non-trusted downloading. Multiple images can be downloaded during one session, which is determined by the number of images described in the non-trusted image module as well as support from the target software.

8.1.6.1 USB Port

The default USB configuration is the differential USB 1.1 client. (The USB 2.0 client is available only for the Tavor P and the PXA300/PXA310 processor platforms.) This port is configured after a power-on reset and is run in interrupt mode. Contact your Marvell Applications Engineer for more information about the communication protocol used by the target and host.

8.1.6.2 UART Port

The default UART configuration is the FFUART. This port is configured after a power-on reset and is run in interrupt mode. Contact your Marvell Applications Engineer for more information about the communication protocol used by the target and host.

8.1.7 Preprogrammed Flash Requirements

For large-volume manufacturing, preprogramming of flash is supported; when using the non-trusted image module, you must:

- Program the non-trusted image module to the correct offset as described in Section 8.1.1, NAND Flash, on page 81, Section 8.1.2, XIP Flash on Chip Select 2, on page 81, and Section 8.1.3, XIP Flash on Chip Select 0, on page 82.
- Program the OEM boot module and any other image described in the non-trusted image module, to the address indicated by the `FlashEntryAddr` field of the non-trusted image module.

Programming of the fuses is optional when the non-trusted image module is used. The Trusted Boot ROM probes the flash memory, searching for the non-trusted image module. After it is found, the image is loaded, the CRC is verified, and then control is transferred.

8.2 Operation Without a Non-Trusted Image Module

The platform can operate without the use of a non-trusted image module. Certain restrictions apply to the image size and execution address, which may also affect the reuse of images on a trusted platform. The fuses on the platform must also be programmed to boot correctly. Contact your Marvell Applications Engineer for more information about fuse programming, which may create the need for multiple part numbers for addressing different flash devices.

8.2.1 NAND Flash

The Trusted Boot ROM supports booting from x8 or x16 NAND devices attached to Chip Select 0 of the processor data flash controller. There is support for both large- and small-block devices; contact your Marvell Applications Engineer for information about specific devices.

The OEM boot image is expected to be loaded to Block 0 starting at offset 0x0. The Trusted Boot ROM reserves the last 24 pages of Block 0 for the relocation table, setting a maximum non-trusted image module size of: (NAND block size - 24 x NAND page size). For example, on a small-block device with a page size of 512 bytes and a block size of 16 KB:

16 KB - (512 bytes x 24) = 4 KB for the maximum size of the non-trusted image module

The platform must also have the platform state fuses programmed correctly for NAND operation. To program the fuses, perform a provisioning step during the manufacturing of the device or use multiple part numbers with the fuses preprogrammed at Marvell.

The Trusted Boot ROM loads the entire contents of Block 0 (minus the last page, which is reserved for the relocation table) into the internal SRAM at location 5C01_3000, if the non-trusted image module has not been found. Because the Trusted Boot ROM enables error-correcting code on the data flash controller, the image must be padded before programming to the NAND device. If the image is not padded, errors may occur on unprogrammed pages and cause the boot operation to halt.

8.2.2 XIP Flash on Chip Select 2

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 2 of the processor static memory controller. There is support for several XIP devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image must be loaded to offset 0x0 in the XIP flash device. The XIP device attached to Chip Select 2 is memory-mapped to 0x1000_0000; this is where the image must be programmed in the flash.

The Trusted Boot ROM jumps to 0x1000_0000 in the XIP device when the platform fuses have been programmed and the non-trusted image module has not been found. To program the fuses, perform a provisioning step during manufacturing of the device or use multiple part numbers with the fuses preprogrammed by Marvell.

8.2.3 XIP Flash on Chip Select 0

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 0 of the processor static memory controller. There is support for several XIP devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image must be loaded to offset 0xC000 in the XIP flash device. The XIP device attached to Chip Select 0 is memory-mapped to 0x0000_0000. Therefore, the image must be programmed to 0x0000_C000.

The Trusted Boot ROM jumps to 0x0000_C000 in the XIP device when the platform fuses have been programmed and the non-trusted image module has not been found. To program the fuses, perform a provisioning step during manufacturing of the device or use multiple part numbers with the fuses preprogrammed by Marvell.

8.2.4 OneNAND Flash

The Trusted Boot ROM supports booting from a OneNAND device attached to Chip Select 2 of the processor static memory controller. There is support for large block devices; contact your Marvell Applications Engineer for information about specific devices.

The OEM boot image is expected to be loaded to Block 0 starting at offset 0x0. The Trusted Boot ROM reserves the last 24 pages of block 0 for the relocation table; this sets a maximum non-trusted image module size of: (OneNAND block size - 24 x NAND page size).

The platform must also have the platform state fuses programmed correctly for OneNAND operation. To program the fuses, perform a provisioning step during the manufacturing of the device or use multiple part numbers with the fuses preprogrammed by Marvell.

The non-trusted image must be loaded to offset 0x0 in the OneNAND flash device. The OneNAND device attached to Chip Select 0 is memory-mapped to 0x1000_0000; this is where the image must be programmed in the flash.

The Trusted Boot ROM jumps to 0x1000_0000 in the OneNAND device when the platform fuses have been programmed and the non-trusted image module has not been found. To program the fuses, perform a provisioning step during the manufacturing of the device or use multiple part numbers with the fuses preprogrammed by Marvell.

8.2.5 MSystems Flash

The Trusted Boot ROM supports booting from an MSystems device attached to Chip Select 2 of the processor static memory controller. There is support for large-block devices; contact your Marvell Applications Engineer for information about specific devices.

The non-trusted image must be loaded to offset 0x0 in the MSystems flash device. The MSystems device attached to Chip Select 0 is memory-mapped to 0x1000_0000. This is where the image must be programmed in the flash.

The Trusted Boot ROM jumps to 0x1000_0000 in the MSystems device when the platform fuses have been programmed and the non-trusted image module has not been found. To program the fuses, perform a provisioning step during the manufacturing of the device or use multiple part numbers with the fuses preprogrammed by Marvell.

8.2.6 Preprogrammed Flash Requirements

The following requirements must be met for preprogrammed devices to boot properly in non-trusted mode:

- The platform fuses must be programmed prior to normal operation by downloading a special provisioning image during the manufacturing process, or by ordering preprogrammed parts from Marvell. Contact your Marvell Applications Engineer for more information about the provisioning steps required to program fuses or for ordering information about preprogrammed parts.
- The image must be programmed at the address offset in flash memory, based on the descriptions provided in:
  - Section 8.1.1, NAND Flash, on page 81
  - Section 8.1.2, XIP Flash on Chip Select 2, on page 81
  - Section 8.1.3, XIP Flash on Chip Select 0, on page 82

- Section 8.1.4, Samsung OneNAND* Flash, on page 82
- Section 8.1.5, SanDisk* Flash, on page 83

Note any restrictions or requirements for each flash device. Verification of the image in flash is the responsibility of the OEM; no image check or verification is performed when the image is programmed.

9 Trusted Boot Operation

The following sections provide information about trusted boot operations, including using a trusted image module, trusted image module validation, flash support, image downloading, preprogrammed flash requirements, and JTAG reenablement.

9.1 Trusted Boot Usage Cases

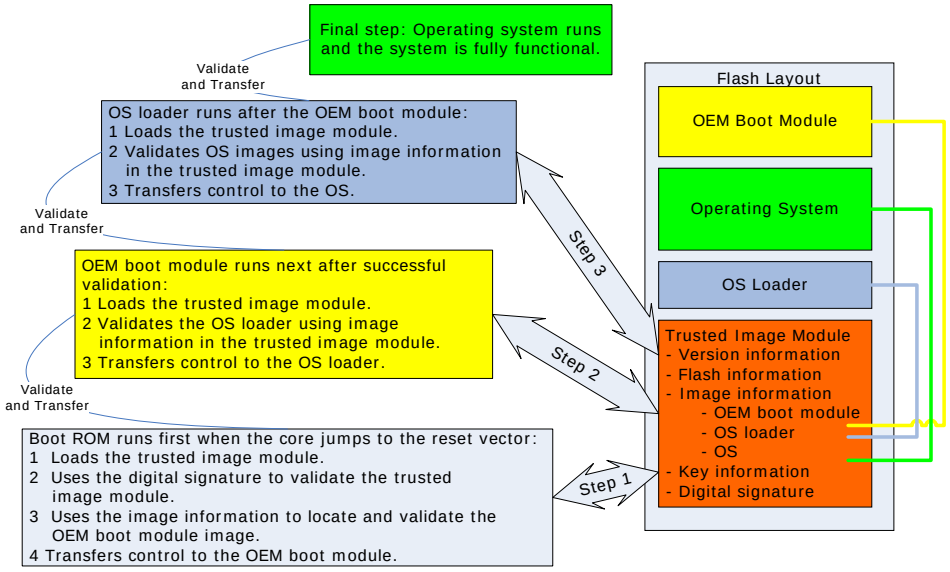
The trusted boot solution is based on the principle of the “chain of trust”, also referred to as layered trust (see Figure 8, “Trusted Boot Operation Using the Trusted Image Module). The Marvell® Trusted Boot ROM is responsible only for securely transferring control to the next layer of software, which is the OEM boot module. The Boot ROM authenticates and checks the integrity of the image with the OEM’s platform public verification key. The OEM boot module establishes an OEM’s proprietary layered trust model to securely validate all of the OEM’s supplementary binaries (the OS, applications, data, and so on), as well as the mobile operator’s service provisioning and downloads.

The layered solution is implemented using the trusted image module defined in Section 7 "Trusted Image Module" on page 71. The trusted image module holds the security information for some or all of the images loaded into the boot flash device. The trusted image module provides a flexible mechanism for trusted boot operations using industry-standard RSA and SHA-1 operations.

Each layer of software, starting with the Boot ROM, uses the information located in the trusted image module to validate one or more images. The implementation defines the number of images to include in the trusted image module for validation, and which layer of software does the validation. Unauthorized modifications to the system software are identified and prevented from running on the system.

The trusted image module is located in the flash memory, according to the usage models defined in the following sections. All other image locations are defined in the trusted image module.

Figure 8: Trusted Boot Operation Using the Trusted Image Module



9.1.1 Trusted Image Module Validation

The trusted image module is a required component for the Marvell® Wireless Trusted Platform (Marvell WTP). The Boot ROM uses the trusted image module to locate and validate all other images in the flash memory that require validation. It also locates state information needed for initializing the Wireless Trusted Module (WTM) after a power state change.

When the Boot ROM runs, it locates the trusted image module based on the usage cases described below. After it is found, it is validated as follows:

1. The SHA-1 hash is calculated using all information up to but excluding the digital signature located in the `RSADigS[64]` field.
2. The digital signature is decrypted using the WTM to get the encrypted hash value.
3. The calculated hash is compared to the encrypted hash.

After the validation of the trusted image module, the Trusted Boot ROM uses the information in the trusted image module to complete the platform validation and setup. The Boot ROM uses the WTM to compare the OEM computed hash of the OEM binding key to the hash stored in the Marvell WTM fuses, during the provisioning stage. If the hashes match, the trusted keys for the platform are loaded from the trusted image module to the WTM. The following platform keys are required:

- JTAG Re-enable Key – Re-enables the JTAG port on the processor. If the flash device has a one-time programmable register, the hash of this key is stored in this register and in the trusted image module.
- Platform Binding Key – Binds the platform to the processor. The hash of this key is programmed into the WTM fuses at the provisioning stage.
- Trusted Certificate Authority Key – Performs as an OEM authorization key to enable certain secure functions on the platform.

9.1.2 NAND Flash

The Trusted Boot ROM supports booting from x8 or x16 NAND devices attached to Chip Select 0 of the processor data flash controller. Large- and small-block devices are supported; contact a Marvell Applications Engineer for information about specific devices.

The trusted image module is expected to be located in Block 0 at offset `0x0` of the NAND device. The Trusted Boot ROM loads the first page of Block 0 and searches for the `"TIMH"` identifier embedded in the version information of the trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point on, the Trusted Boot ROM uses the trusted image module to load and validate the OEM boot module. If the trusted image module is not found, an error condition is reported and the boot operation halts.

The OEM boot module is described by the image information contained in the “IMAGE INFORMATION” array. It is identified by the `"OBMI"` image identifier, which is a required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

After the image has been loaded to the correct address, the image must be validated. The hash of the OEM boot module is calculated using the WTM. The `ImageSizeToHash` field determines how much of the image was used in the SHA-1 hash calculation. The hash calculated by the WTM is then compared to the hash stored in the `Hash[8]` array field. If the hashes match, control is transferred to the OEM boot module.

9.1.3 XIP Flash on Chip Select 2

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 2 of the processor static memory controller. Several XIP devices are supported; contact your Marvell Applications Engineer for questions about specific devices.

The trusted image module is expected to be located at offset `0x0` of the XIP device. For Chip Select 2, the XIP device is memory-mapped to `0x1000_0000`; this is the address where the trusted image module must reside. The Trusted Boot ROM searches for the `"TIMH"` identifier embedded in the version information of the trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point, the Trusted Boot ROM uses the trusted image module to load the OEM boot module. If the trusted image module is not found, an error condition is reported and the boot operation halts.

The OEM boot module is described by the image information contained in the image information array. It is identified by the `"OBMI"` image identifier, which is a required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The `ImageSize` entry determines the number of bytes that are loaded.

After the image has been loaded to the correct address, the image must be validated. The hash of the OEM boot module is calculated using the WTM. The `ImageSizeToHash` field determines how much of the image was used in the SHA-1 hash calculation. The hash calculated by the WTM is then compared to the hash stored in the `Hash[8]` array field. Upon a successful compare operation, control is transferred to the OEM boot module.

9.1.4 XIP Flash on Chip Select 0

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 0 of the processor static memory controller. Several XIP devices are supported; contact your Marvell Applications Engineer for questions about specific devices.

The trusted image module is expected to be located at offset `0x0` of the XIP device. For Chip Select 0, the XIP device is memory-mapped to `0x0000_0000;` the address where the trusted image module must reside is `0x0000_C000`. The Trusted Boot ROM searches for the `"TIMH"` identifier

embedded in the version information of the trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point, the Trusted Boot ROM uses the Trusted Image Module to load the OEM boot module. If the Trusted Image module is not found, an error condition is reported and the boot operation halts.

The OEM boot module is described by the image information contained in the “IMAGE INFORMATION” array. It is identified by the `"OBMI"` image identifier, which is a required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The `ImageSize` entry determines the number of bytes that are loaded.

After the image has been loaded to the correct address, the image must be validated. The hash of the OEM boot module is calculated using the WTM. The `ImageSizeToHash` field determines how much of the image was used in the SHA-1 hash calculation. The hash calculated by the WTM is then compared to the hash stored in the `Hash[8]` array field. Upon a successful compare operation, control is transferred to the OEM boot module.

9.1.5 Samsung OneNAND™ Flash

The Trusted Boot ROM supports booting from an x16 OneNAND device attached to Chip Select 2 of the processor static memory controller. Large-block devices are supported; contact your Marvell Applications Engineer for questions about specific devices.

The trusted image module is expected to be located in Block 0 at offset `0x0` of the OneNAND device. The OneNAND device is memory-mapped to `0x1000_0000`. The Trusted Boot ROM loads the first page of Block 0 and searches for the `"TIMH"` identifier embedded in the version information of the trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point on, the Trusted Boot ROM uses the trusted image module to load and validate the OEM boot module. If the trusted image module is not found, an error condition is reported and the boot operation halts.

The OEM boot module is described by the image information contained in the “IMAGE INFORMATION” array. It is identified by the `"OBMI"` image identifier, which is a required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

After the image has been loaded to the correct address, the image must be validated. The hash of the OEM boot module is calculated using the WTM. The `ImageSizeToHash` field determines how much of the image was used in the SHA-1 hash calculation. The hash calculated by the WTM is then compared to the hash stored in the `Hash[8]` array field. If the hashes match, control is transferred to the OEM boot module.

Section 5.5 "OneNAND Support" on page 43 specifies the Device ID’s and accompanying stepping generation codes that are supported.

9.1.6 SanDisk* Flash

The Trusted Boot ROM supports booting from the x16 SanDisk device attached to Chip Select 2 of the processor static memory controller. Large-block devices are supported; contact your Marvell Applications Engineer for information about specific devices.

The trusted image module is expected to be located in Block 0 at offset `0x0` of the device. The device is memory-mapped to `0x1000_0000`. The Trusted Boot ROM loads the first page of Block 0 and searches for the `"TIMH"` identifier embedded in the version information of the trusted image module. If the structure is found, it is loaded into the internal SRAM of the system. From this point on, the Trusted Boot ROM uses the trusted image module to load and validate the OEM boot module. If the trusted image module is not found, an error condition is reported and the boot operation halts.

The OEM boot module is described by the image information contained in the “IMAGE INFORMATION” array. It is identified by the `"OBMI"` image identifier, which is a required identifier for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by `FlashEntryAddr` to the location pointed to by `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

After the image has been loaded to the correct address, the image must be validated. The hash of the OEM boot module is calculated using the WTM. The `ImageSizeToHash` field determines how much of the image was used in the SHA-1 hash calculation. The hash calculated by the WTM is then compared to the hash stored in the `Hash[8]` array field. If the hashes match, control is transferred to the OEM boot module.

The mDOC device acts as a NAND disk; however, no file system is mounted during the boot operation for PXA300/PXA310 processors or the Tavor P processor. Instead, the mDOC driver code in the Boot ROM accesses the mDOC device via formatted partitions and sector offsets within the partitions. The trusted and non-trusted image modules must reside in Partition 2, Sector Offset 0. Any other images such as the OEM boot module must reside in Partition 2 or greater. Partition 1 is an one-time programmable (OTP) partition that stores keys for trusted boot operations, but is not otherwise usable. mDOC sectors are 512 bytes.

The 32-bit flash address for mDOC images are calculated as follows:

28-bit sector offset	4-bit partition #

Flash positions within the mDOC H3 flash device for use with input files for the Trusted Boot Builder tool are calculated as follows:

mDOC address:

TIM (must be in Partition 2, Sector Offset 0) = 0x2

OEM boot module in Partition 5 starting at Sector Offset 32 = 0x205

OEM boot module in Partition 3 starting at Sector Offset 506 = 0x1FA3

9.1.7 Image Downloading

The Trusted Boot ROM enables the differential USB 1.1 client and the FFUART port shortly after a power-on reset. An image can be downloaded over one of these ports using the Marvell `WTPTP.exe` utility described in Section 12.2 "Download Tools" on page 103. The use of a trusted image module is required for trusted downloading. Multiple images can be downloaded during one session; this is determined by the number of images described in the trusted image module, as well as support from the target software.

Contact your Marvell Applications Engineer for more information about using the Marvell-supplied tools for creating the trusted image module and downloading images to the target.

9.1.7.1 USB Port

The default USB configuration is the differential USB 1.1 client. This port is configured after a power-on reset and is run in interrupt mode. Contact your Marvell Applications Engineer for more information about the communication protocol used by the target and host.

9.1.7.2 UART Port

The default UART configuration is FFUART. This port is configured after a power-on reset and is run in interrupt mode. Contact your Marvell Applications Engineer for more information about the communication protocol used by the target and host.

9.2 Preprogrammed Flash Requirements

For large-volume manufacturing, the preprogramming of flash memory is supported. When using a trusted image module, the requirements are as follows:

- Program the trusted image module to the correct offset; contact your Marvell Applications Engineer for more information.
- Program the OEM boot module and any other image described in the non-trusted image module, to the address indicated by `FlashEntryAddr` of the trusted image module.
- Perform device provisioning using a Device Keying Binary.

Programming of the fuses is required when a trusted image module is used. The Trusted Boot ROM examines flash memory and searches for the non-trusted image module. After it is found, the image is loaded, the CRC verified, and control is transferred.

9.3 JTAG Re-enablement

To troubleshoot failed parts returned from the field, it is necessary to gain access to the device through the JTAG port. However, access to this port is disabled by non-volatile fuses in devices with the security module enabled. This feature protects the security module and the device from compromise, but also prevents the primary debug interface. However, there is a challenge/response mechanism for re-enabling the JTAG port.

After the JTAG port is re-enabled, it remains accessible until the next power-on reset. After a power-on reset, the JTAG challenge/response must occur again before the JTAG port can be used. The Boot ROM enables the JTAG challenge/response mechanism during initialization of the WTM, using the OEM's JTAG re-enabling key hash stored in the trusted image module and in the one-time programmable registers (if available) during the device keying process.

After the WTM is enabled for the JTAG challenge/response, one attempt per power-on reset is allowed. After a failed attempt, the WTM locks out the JTAG port until a power-on reset has occurred. If the validation of the trusted image module fails during the platform initialization, a new trusted image module must first be downloaded and validated using the OEM platform bind key burned into the WTM fuses.

10 TIM/NTIM Support For Memory Devices

This section provides details on memory device support for trusted and non-trusted boot.

TIM - Trusted Image Module

NTIM - Non-Trusted Image Module

10.1 NAND Flash

The Marvell® Trusted Boot ROM supports booting from x8 or x16 NAND devices attached to Chip Select 0 of the processor data flash controller. Both large and small block devices are supported. Contact your Marvell FAE with questions about specific devices.

The TIM/NTIM is located in Block 0 at offset `0x0` of the NAND device. The Trusted Boot ROM loads the first page of Block 0 and searches for the "`TIMH`" identifier embedded in the version information of the Non-Trusted Image Module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the Non-Trusted Image Module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array and is identified by the "`OBMI`" image identifier. This required identifier is necessary for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by the `FlashEntryAddr` to the location pointed to by the `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

Once the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the `ImageSize`, `LoadAddr`, and `ImageSizeToCRC` entries. Then, the calculated CRC is compared to the CRC stored in the Non-Trusted Image Module in the CRC field. Upon a successful check, control is transferred to the image at the load address.

10.2 XIP Flash on Chip Select 2

The Trusted Boot ROM supports booting from an XIP device attached to Chip Select 2 of the processor static memory controller. Several XIP devices are supported so contact your Marvell FAE with questions about specific devices.

The TIM/NTIM is located at offset `0x0` of the XIP device. For Chip Select 2, the XIP device is memory mapped to `0x1000_0000`, and is the address where the Non-Trusted Image Module must reside. The Trusted Boot ROM searches for the "`TIMH`" identifier embedded in the version information of the TIM/NTIM. If the structure is found, it is loaded into the internal SRAM of the system. From this point on, the Trusted Boot ROM uses the TIM/NTIM to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array. It is identified by the "`OBMI`" image identifier. This required identifier is necessary for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by the `FlashEntryAddr` to the location pointed to by the `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

Once the image has been loaded to the correct address, the CRC `ImageSizeToCRC` must be verified. The image CRC is calculated based on the `ImageSize` and `LoadAddr` entries. Then, the calculated CRC is compared to the CRC stored in the Non-Trusted Image Module in the CRC field. Upon a successful check, control is transferred to the image at the load address.

10.3 OneNAND Flash

The Trusted Boot ROM supports booting from a x16 OneNAND device attached to Chip Select 2 of the processor static memory controller. Refer to Section 5.2, “Boot ROM NAND Device Support” for supported NAND devices.

The Non-Trusted Image Module is located in Block 0 at offset `0x0` of the OneNAND device. The OneNAND device is memory mapped to `0x1000_0000`. The Trusted Boot ROM loads the first page of Block 0 and searches for the “`TIMH`” identifier embedded in the version information of the Non-Trusted Image Module. If the structure is found, it is loaded into the internal SRAM of the system. From this point forward, the Trusted Boot ROM uses the Non-Trusted Image Module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array, and is identified by the “`OBMI`” image identifier. This required identifier is necessary for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by the `FlashEntryAddr` to the location pointed to by the `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

Once the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the `ImageSize`, `LoadAddr`, and `ImageSizeToCRC` entries. Then, the calculated CRC is compared to the CRC stored in the Non-Trusted Image Module in the CRC field. Upon a successful check, control is transferred to the image at the load address.

Refer to Section 5.5, “OneNAND Support” for more details.

10.4 SanDisk Flash

The Trusted Boot ROM supports booting from a x16 SanDisk device attached to Chip Select 2 of the processor static memory controller. Large block devices are supported so contact your Marvell FAE with questions about specific devices.

The Non-Trusted Image Module is located in Block 0 at offset `0x0` of the SanDisk device, which is memory mapped to `0x1000_0000`. The Trusted Boot ROM loads the first page of Block 0 and searches for the “`TIMH`” identifier embedded in the version information of the Non-Trusted Image Module. If the structure is found, it is loaded into the internal SRAM of the system. From this point on, the Trusted Boot ROM uses the Non-Trusted Image Module to load the OEM boot module.

The OEM boot module is described by the image information contained in the image information array, and is identified by the “`OBMI`” image identifier. This required identifier is necessary for proper use with the Trusted Boot ROM. Using the information that describes the OEM boot module, the image is loaded from the flash offset pointed to by the `FlashEntryAddr` to the location pointed to by the `LoadAddr`. The number of bytes loaded is determined by the `ImageSize` entry.

Once the image has been loaded to the correct address, the CRC must be verified. The image CRC is calculated based on the `ImageSize`, `LoadAddr`, and `ImageSizeToCRC` entries. Then, the calculated CRC is compared to the CRC stored in the Non-Trusted Image Module in the CRC fie.

Refer to Section 5.6, “mDOC Support” for more details.

11 Communication Protocol

The chapter describes the relevant details of the USB/UART communication protocol to allow OEMs to port their existing proprietary USB/UART applications to support communication with the Boot ROM.

The communication protocol is used to download images during the device keying process, as well as for the JTAG re-enabling process.

Refer to Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 for specific processor support of this feature.

In this section, the "`Host`" refers to the `WTPTool.exe` application and "`Target`" refers to the Boot ROM.

The communication protocol follows a strict handshaking methodology, which is always initiated by the host. The host sends a command packet and the target responds with a status packet (response packet).

Figure 9 shows the process flow for the JTAG re-enablement sequence. Figure 10 shows the process flow for the download sequence.

Figure 9: JTAG Re-enable Flow Diagram

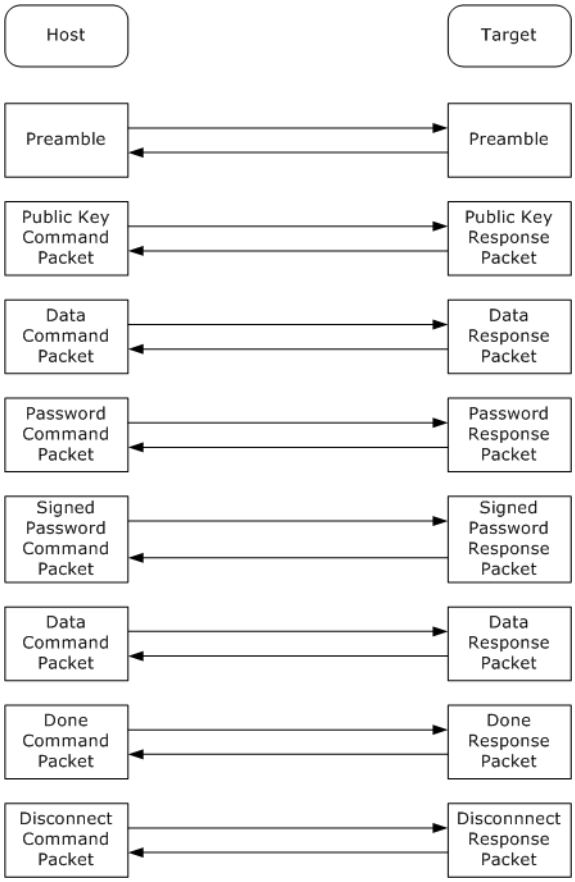
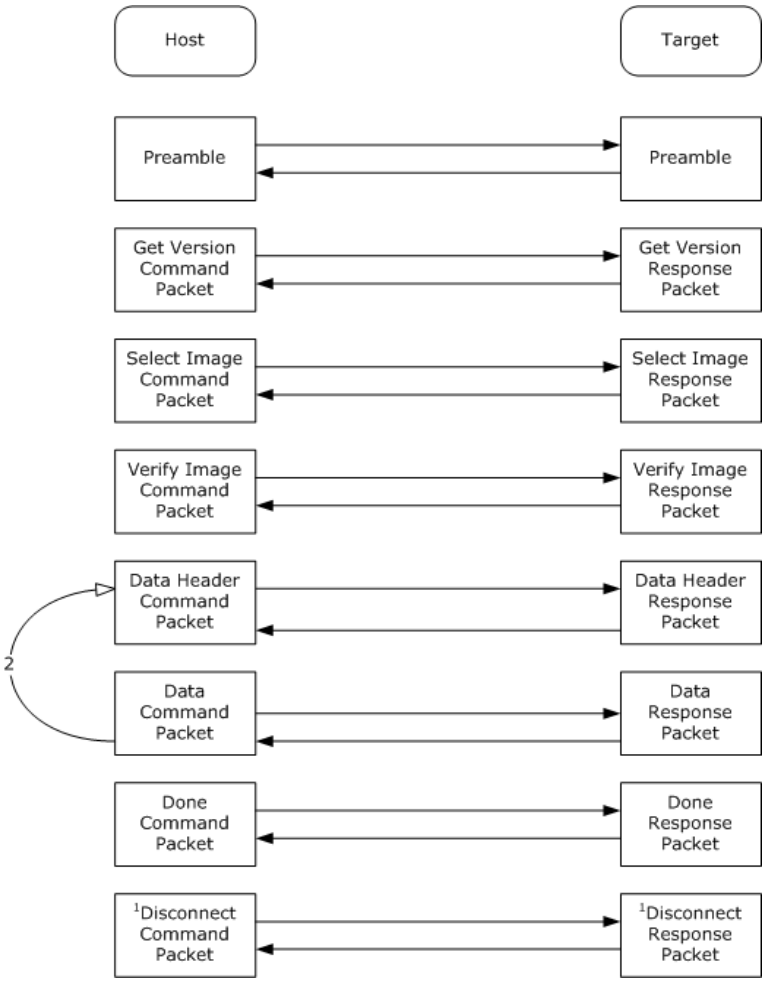


Figure 10: Download Flow Diagram



NOTE: 1. The disconnect command is only issued after the target has transmitted all of the files.
2. The data header and data command/response packets are sent continually until all data has been transmitted.

11.1 Preamble

The preamble data stream is a four-byte data packet containing 0x00, 0xD3, 0x02, and 0x2B. Table 41 represents a 32-bit word: 0x2B02D300. The preamble data stream requires that the bytes are in network byte ordering.

Table 41: Preamble

Byte-3	Byte-2	Byte-1	Byte-0
0x2B	0x02	0xD3	0x00

The target responds to the preamble from the host with the same preamble.

11.2 Structure for Host Commands

The structure of all commands sent by the host follows this format:

```
struct Command
{
   Byte   CMD
   Byte   SEQ
   Byte   CID
   Byte   Flags
   Unsigned intLEN
   Byte [LEN]Data
}
```

CMD (Command) – Contains the opcode that indicates the type of command being sent. The size is one byte.

SEQ (Sequence) – Used during data transmission (when the data command is used) to ensure that the block of data that the host sends matches the block of data that the target is expecting. The sequence number is 0 for all other commands. The sequence number is 1 for the first data transmission, 2 for the second, and so on. Since the size of the sequence field is 1 byte, the sequence number rolls over after 255 data transmissions.

CID (Command ID) – Specific number that relates all of the commands (and responses) of a single flow. A flow is the communication from the preamble to the done acknowledgement. The host defines the CID when it sends the first command after the preamble. The same CID is used until the done command after a download or a JTAG reenablement. If another download follows, the host must generate a new CID for the next download flow (after the next preamble).

Flags – Bits [7:1] reserved.

- Bit 0 – Endian format of the data. Once set, this flag must remain the same throughout the flow.
  - 1: big endian
  - 0: little endian

LEN (Data Length in Bytes) – Number of bytes of the data field in the current command. This length does not include the CMD, SEQ, CID, Flags, or LEN fields. It is the total length (in bytes) of the data in the data field only. The LEN field itself is 4 bytes long, and is in little endian format.

Data – Data field associated with the current command. The number of bytes of this field must equal the LEN value above. If LEN is zero, then this field does not exist. On a word (32-bit) basis, the default configuration is to send the data in little endian format.

11.3 List of Commands

Table 42 lists all commands sent by the host.

Table 42: Host Commands

Commands	CMD	SEQ	LEN	Data	Comment
Public Key	0x24	0	0	None	Indicates that the next command is a data command containing the public key
Password	0x28	0	0	None	Tells the target to send a 64-bit password
Signed Password	0x25	0	0	None	Indicates that the next command is a data command containing the signed password
Get Version	0x20	0	0	None	Tells the target to send the version information
Select Image	0x26	0	0	None	Tells the target to respond with the image type to be downloaded
Verify Image	0x27	0	1	0 = ACK 1 = NACK	Tells the target whether the image type asked for in Select Image is available
Data Header	0x2a	y	4	Size	Tells the target how much data is left to be downloaded
Data	0x22	y	x	Data	Sends the target the next block of data
Message	0x2b	0	0	None	Tells the target to send its message
Done	0x30	0	0	None	Tells the target that the current flow is complete, yet more images are available for download
Disconnect	0x31	0	0	None	Tells the target that the current flow is complete and there are no more images left to download

x: `LEN` value is variable: `SEQ` number is incremental

Note

Note

The `CID` is not listed because it is unique to each flow.

11.4 Structure of Status Responses

The structure of all status responses sent by the target follows this format:

```
struct Status
{
   ByteCMD
   ByteSEQ
   ByteCID
   ByteStatus
   ByteFlags
   ByteLEN
   Byte[Len]Data
}
```

`CMD` – Same opcode as the command this response packet is acknowledging.

SEQ (Sequence) – Used during data transmission in response to a data command to keep the host and target in synchronization. The sequence number is 1 for the first data transmission, 2 for the second, and so on. Since the size of the sequence field is 1 byte, the sequence number rolls over after 255 data transmissions.

CID (Command ID) – Specific number that relates all of the commands (and responses) of a single flow. A flow is the communication from the preamble to the “done” acknowledgement. The host defines the CID when it sends the first command after the preamble. The same CID is used until the done command after a download or a JTAG re-enablement. If another download follows, the host must generate a new CID for the next download flow (after the next preamble).

Status – Status code of the target in response to the last command sent by the host.

Flags – Bits [7:2] reserved.

- Bit 0 – Message Flag. Tells the host that the target wants to send a message. The next command the host should send is a message command. The target lowers this flag when no messages remain in the queue.
  - 1: message waiting to be sent
  - 0: no messages
- Bit 1 – Message Type. This flag is applicable only when sent in a message response packet (the response packet CMD is 0x2B). This flag tells the target whether data in the data field is an ASCII string or an integer value representing an error code.
  - 1: integer error code
  - 0: ASCII string

For additional information about messaging, see Section 11.6, “Messages”.

LEN (Data Length in Bytes) – Size of the data field of the current response. It is the total length (in bytes) of the data in the data field only. The maximum value of LEN is 255 bytes.

Data – Data field associated with the current response. The number of bytes of this field must equal the LEN value above. If LEN is zero, then this field does not exist. On a word (32-bit) basis, this data is in little endian format. The maximum size of the data field is 255 bytes.

11.5 Responses

Every command sent by the host requires the target to respond with a status packet. Some of the responses require data in the data field while others do not. Table 43 describes the contents of the data field for each response packet.

Table 43: Target Responses

Commands	CMD	LEN	Data
Public Key	0x24	0	No data needed
Password	0x28	8	A 64-bit password
Signed Password	0x25	0	No data needed
Get Version	0x20	12	The version information. First 4 bytes are ASCII characters and represent the target stepping version. The second 4 bytes is an integer capturing the date. The last 4 bytes are ASCII characters and represent the type of processor.
Select Image	0x26	4	Image Identifier
Verify Image	0x27	0	No data needed

Table 43: Target Responses (Continued)

Commands	CMD	LEN	Data
Data Header	0x2a	4	A 32-bit integer that tells the host how much data to send in the next Data command
Data	0x22	0	No data needed
Message	0x2b	x	ASCII string. This is a message that the target wants printed for the user.
Done	0x30	0	No data needed
Disconnect	0x31	0	No data needed

11.6 Messages

At any time during the communication process, the target may send a text message to the host by the target raising Bit 0 of the flag field. The host should then send the message command as the following command.

Note

Note

The host is not required to send the message command as soon as the message flag has been raised. The target keeps the message in the queue and message flag bit raised until the message command is sent and the message is handled.

11.7 Disconnect

After the target has finished downloading all of the images, the host issues the disconnect command. The target does not respond to the command until it has finished its operations, which allows the target to fill up the message queue with any messages needed to be sent to users.

Once the target issues the response packet to the disconnect command, the host must check the message flag. If the flag is not set, the host shuts down and the target transfers control. However, if the flag is set, the host must continue issuing message commands until the message flag is lowered. The host should ignore the status field during this sequence.

11.8 Status Codes

Table 44 describes the current status codes communicated back to the host application.

Table 44: Status Codes

Error Code	Description
0x00	ACK
0x01	NACK
0x02	Sequence error

12 Host Tools

Marvell provides a sample tool package that can be used as a starting base to create OEM-specific tools. Contact a Marvell field representative for information on the Marvell® Wireless Trusted Platform Tool Package.

Note

Note Marvell also provides additional documentation on the Wireless Trusted Platform Tool Package. The Porting Guide and the Users Guide offer more detail on the software components and build information.

12.1 Trusted Image Tools

The step in enabling trusted boot capability is the generation of trusted images, which requires the generation of the integrated verification module using the OEM keys. The integrated verification module and binary are then packaged together. Once the image is packaged with the integrated verification module, a digital signature must be created and stored in the integrated verification module. This creates a binary image ready to be downloaded or burnt to flash.

12.2 Download Tools

The Boot ROM supports downloading via USB or FFUART. See Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 for platform specific requirements. The protocol for downloading is described in Chapter 7, “Trusted Image Module.” The Intel* Intelligent Cellular Analysis Tool utility can also be used to download images, but not for JTAG re-enabling.

12.3 JTAG Re-enable Tools

Tools for JTAG re-enabling are required to perform the challenge/response mechanism. These tools must have access to the OEM signing keys used to device key the platform. These keys are used to encrypt a random password and return it for verification.

13 Other Boot ROM Features

This chapter provides details about the functionality between the Boot ROM and other system components.

13.1 Optional Settings in the TIM/NTIM Modules

The reserved areas of the trusted image module and non-trusted image module are provided for use by the OEM. A few optional settings are available to modify certain functionality of the Boot ROM, specifically, settings for an optional download port and custom GPIO settings. The structure below is interpreted by the Boot ROM and the settings are used at boot time or any other time the Boot ROM owns system resources. Refer to 5.8 "Handling Power Mode and Reset Transitions" to see the list of resets and power modes.

```
typedef struct
{
UINT_T Addr;
UINT_T Value;
}GPIO_DEF, *pGPIO_DEF;
```

```
typedef struct
{
UINT_T Identifier;
UINT_T PortType;
UINT_T Port;
UINT_T GPIOPresent;
UINT_T NumGpios;
pGPIO_DEFGPIO;
}OPT_SET, *pOPT_SET
```

```
//Option Identifiers
#define OPTIONALHEADER        0x4F505448                // "OPTH"
#define FFIDENTIFIER          0x00004646                // "FF"
#define ALTIDENTIFIER         0x00414C54                // "ALT"
#define DIFFIDENTIFIER        0x44696666                // "Diff"
#define SEIDENTIFIER          0x00005345                // "SE"
#define UARTID                0x55415254                // "UART"
#define USBID                 0x00555342                // "USB"
#define PINSIDENTIFIER        0x50696E73                // "Pins"
#define TERMINATOR            0x5465726D                // "Term"
```

13.2 Tamper Recovery Mechanisms

A tamper recover mechanism was implemented in the Boot ROM for trusted platforms. Refer to Table 1, Version 2.xx and Version 3.xx High Level Differences, on page 15 for specific processor support of this feature.

The tamper recover mechanism works with the Device Keying Binary and Wireless Trusted Platform Service Package (WTPSP) security device driver. An install status word has been added to the Wireless Trusted Module (WTM) save state data that allows the Boot ROM, the Device Keying Binary, and WTPSP to keep track of the status of the last boot attempt.

Three states are defined: successful boot, backup boot, and corrupted save state. If the state is anything but a successful boot or backup state, the Boot ROM performs a WTM initialization to an

uninitialized state. This step allows the WTPSP driver to notify the OS and correct the problem by creating a new save state file. Two boots are required to correct the tamper state if both the primary and backup copies of the save state are corrupted.

A Return Code Definitions

The following list contains the return codes and definitions.

Table 45: Return Codes and Definitions

/* General Error Code Definitions **/**	**0x0 - 0x1F**
#define NoError	0x0
#define NotFoundError	0x1
#define GeneralError	0x2
#define WriteError	0x3
#define ReadError	0x4
#define NotSupportedError	0x5
#define InvalidPlatformConfigError	0x6
#define PlatformBusy	0x7
#define PlatformReady	0x8
#define InvalidSizeError	0x9
// Flash Related Errors	**0x20 - 0x3F**
#define EraseError	0x20
#define ProgramError	0x21
#define InvalidBootTypeError	0x22
#define ProtectionRegProgramError	0x23
#define NoOTPFound	0x24
#define BBTReadError	0x25
#define MDOCInitFailed	0x26
#define OneNandInitFailed	0x27
#define MDOCFormatFailed	0x28
#define BBTExhaustedError	0x29
#define FlashDriverInitError	0x30
#define FlashFuncNotDefined	0x31
#define OTPError	0x32
#define InvalidAddressRangeError	0x33
// DFC Related Errors	**0x40 - 0x5F**
#define DFCDoubleBitError	0x40

Table 45: Return Codes and Definitions (Continued)

#define DFCSingleBitError	0x41
#define DFCCS0BadBlockDetected	0x42
#define DFCCS1BadBlockDetected	0x43
#defineDFCInitFailed	0x44
// Security Related Errors	**0x60 - 0x8F**
#define InvalidOEMVerifyKeyError	0x60
#define InvalidOBMImageError	0x61
#define SecureBootFailureError	0x62
#define InvalidSecureBootMethodError	0x63
#define UnsupportedFlashError	0x64
#define InvalidCaddoFIFOEntryError	0x65
#define InvalidCaddoKeyNumberError	0x66
#define InvalidCaddoKeyTypeError	0x67
#define RSADigitalSignatureDecryptError	0x68
#define InvalidHashValueLengthError	0x69
#define InvalidTIMImageError	0x6A
#define HashSizeMismatch	0x6B
#define InvalidKeyHashError	0x6C
#define TIMNotFound	0x6D
#define WTMStateError	0x6E
#define FuseRWError	0x6F
#define InvalidOTPHashError	0x70
#define CRCFailedError	0x71
#define SaveStateNotFound	0x72
#define WTMInitializationError	0x73
#define ImageNotFound	0x74
#define InvalidImageHash	0x75
#define MicroCodePatchingError	0x76
#define SetJtagKeyError	0x77
#define WTMDisabled	0x78
// Download Protocols	**0x90 - 0xAF**
#define DownloadPortError	0x90
#define DownloadError	0x91
#define FlashNotErasedError	0x92

Table 45: Return Codes and Definitions (Continued)

#define InvalidKeyLengthError	0x93
#define DownloadImageTooBigError	0x94
#define UsbPreambleError	0x95
#define TimeOutError	0x96
#define UartReadWriteTimeOutError	0x97
#define UnknownImageError	0x98
#define MessageBufferFullError	0x99
#define NoEnumerationResponseTimeOutError	0x9A
#define UnknownProtocolCmd	0x9B
//JTAG ReEnable Error Codes	**0xB0 - 0xCF**
#define JtagReEnableError	0xB0
#define JtagReEnableOEMPubKeyError	0xB1
#define JtagReEnableOEMSignedPassWdError	0xB2
#define JtagReEnableTimeOutError	0xB3
#define JtagReEnableOEMKeyLengthError	0xB4
//SD/MMC Error Codes	**0xD0 - 0xE2**
#define SDMMC_SWITCH_ERROR	0xD0
#define SDMMC_ERASE_RESET_ERROR	0xD1
#define SDMMC_CIDCSD_OVERWRITE_ERROR	0xD2
#define SDMMC_OVERRUN_ERROR	0xD3
#define SDMMC_UNDERUN_ERROR	0xD4
#define SDMMC_GENERAL_ERROR	0xD5
#define SDMMC_CC_ERROR	0xD6
#define SDMMC_ECC_ERROR	0xD7
#define SDMMC_ILL_CMD_ERROR	0xD8
#define SDMMC_COM_CRC_ERROR	0xD9
#define SDMMC_LOCK_ULOCK_ERROR	0xDA
#define SDMMC_LOCK_ERROR	0xDB
#define SDMMC_WP_ERROR	0xDC
#define SDMMC_ERASE_PARAM_ERROR	0xDD
#define SDMMC_ERASE_SEQ_ERROR	0xDE
#define SDMMC_BLK_LEN_ERROR	0xDF
#define SDMMC_ADDR_MISALIGN_ERROR	0xE0
#define SDMMC_ADDR_RANGE_ERROR	0xE1

Table 45: Return Codes and Definitions (Continued)

#define SDMMCDeviceNotReadyError	0xE2
#define SDMMCInitializationError	0xE3
#define SDMMCDeviceVoltageNotSupported	0xE4
Other DFC Errors	
Double Bit Error on Read	0xFFFFFFF6 or 0xF6
DFC Timeout on Read Command	0xFFFFFFF9 or 0xF9
DFC Timeout on Page Read	0xFFFFFFF8 or 0xF8
DFC Timeout on Read ID Command	0xFFFFFFFF or 0XFF

THIS PAGE INTENTIONALLY LEFT BLANK

MARVELL®

MOVING FORWARD FASTER®

Marvell Semiconductor, Inc.

700 First Avenue
Sunnyvale, CA 94089, USA

Tel: 1.408.222.2500
Fax: 1.408.752.9028

www.marvell.com

Worldwide Corporate Offices

Marvell Semiconductor, Inc.
700 First Avenue
Sunnyvale, CA 94089, USA
Tel: 1.408.222.2500
Fax: 1.408.752.9028

Marvell Semiconductor, Inc.
5400 Bayfront Plaza
Santa Clara, CA 95054, USA
Tel: 1.408.222.2500

Marvell Asia Pte, Ltd.
151 Lorong Chuan, #02-05
New Tech Park, Singapore 556741
Tel: 65.6756.1600
Fax: 65.6756.7600

Marvell Japan K.K.
Shinjuku Center Bldg. 44F
1-25-1, Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0644, Japan
Tel: 81.(0).3.5324.0355
Fax: 81.(0).3.5324.0354

Marvell Semiconductor Israel, Ltd.
6 Hamada Street
Mordot HaCarmel Industrial Park
Yokneam 20692, Israel
Tel: 972.(0).4.909.1500
Fax: 972.(0).4.909.1501

Marvell Semiconductor Korea, Ltd.
Rm. 603, Trade Center
159-2 Samsung-Dong, Kangnam-Ku
Seoul 135-731, Korea
Tel: 82.(0).2.551-6070/6079
Fax: 82.(0).2.551.6080

Radlan Computer Communications, Ltd.
Atidim Technological Park, Bldg. #4
Tel Aviv 61131, Israel
Tel: 972.(0).3.645.8555
Fax: 972.(0).3.645.8544

Worldwide Sales Offices

Western US

Marvell
700 First Avenue
Sunnyvale, CA 94089, USA
Tel: 1.408.222.2500
Fax: 1.408.752.9028
Sales Fax: 1.408.752.9029

Marvell
5400 Bayfront Plaza
Santa Clara, CA 95054, USA
Tel: 1.408.222.2500

Central US

Marvell
9600 North MoPac Drive, Suite #215
Austin, TX 78759, USA
Tel: 1.512.343.0593
Fax: 1.512.340.9970

Eastern US/Canada

Marvell
Parlee Office Park
1 Meeting House Road, Suite 1
Chelmsford, MA 01824 , USA
Tel: 1.978.250.0588
Fax: 1.978.250.0589

Europe

Marvell
5 Marchmont Gate
Boundary Way
Hemel Hempstead
Hertfordshire, HP2 7BF
United Kingdom
Tel: 44.(0).1442.211668
Fax: 44.(0).1442.211543

Israel

Marvell
6 Hamada Street
Mordot HaCarmel Industrial Park
Yokneam 20692, Israel
Tel: 972.(0).4.909.1500
Fax: 972.(0).4.909.1501

China

Marvell
5J1, 1800 Zhongshan West Road
Shanghai, PRC 200233
Tel: 86.21.6440.1350
Fax: 86.21.6440.0799

Marvell
Rm. 1102/1103, Jintian Fudi Mansion
#9 An Ning Zhuang West Rd.
Qing He, Haidian District
Beijing, PRC 100085
Tel: 86.10.8274.3831
Fax: 86.10.8274.3830

Japan

Marvell
Shinjuku Center Bldg. 44F
1-25-1, Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0644, Japan
Tel: 81.(0).3.5324.0355
Fax: 81.(0).3.5324.0354

Taiwan

Marvell
2Fl., No.1, Alley 20, Lane 407, Sec. 2
Ti-Ding Blvd., Nei Hu District
Taipei, Taiwan, 114, R.O.C
Tel: 886.(0).2.8177.7071
Fax: 886.(0).2.8752.5707

Korea

Marvell
Rm. 603, Trade Center
159-2 Samsung-Dong, Kangnam-Ku
Seoul 135-731, Korea
Tel: 82.(0).2.551-6070/6079
Fax: 82.(0).2.551.6080

For more information, visit our website at: www.marvell.com