UG0881
User Guide
PolarFire SoC FPGA Booting And Configuration

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

50200881. 2.0 4/20

Contents

Figures

Tables

1 Booting And Configuration

PolarFire SoC FPGAs use advanced power-up circuitry to ensure reliable power on at power-up and reset. At power-up and reset, PolarFire SoC FPGA boot-up sequence follows Power-on reset (POR), Device boot, Design initialization, Microcontroller Subsystem (MSS) pre-boot, and MSS user boot. This document describes MSS pre-boot and MSS User Boot. For information about POR, Device Boot and Design initialization, see *UG0890: PolarFire SoC FPGA Power-Up and Resets User Guide*.

For more information about MSS features, see *UG0880: PolarFire SoC MSS User Guide*.

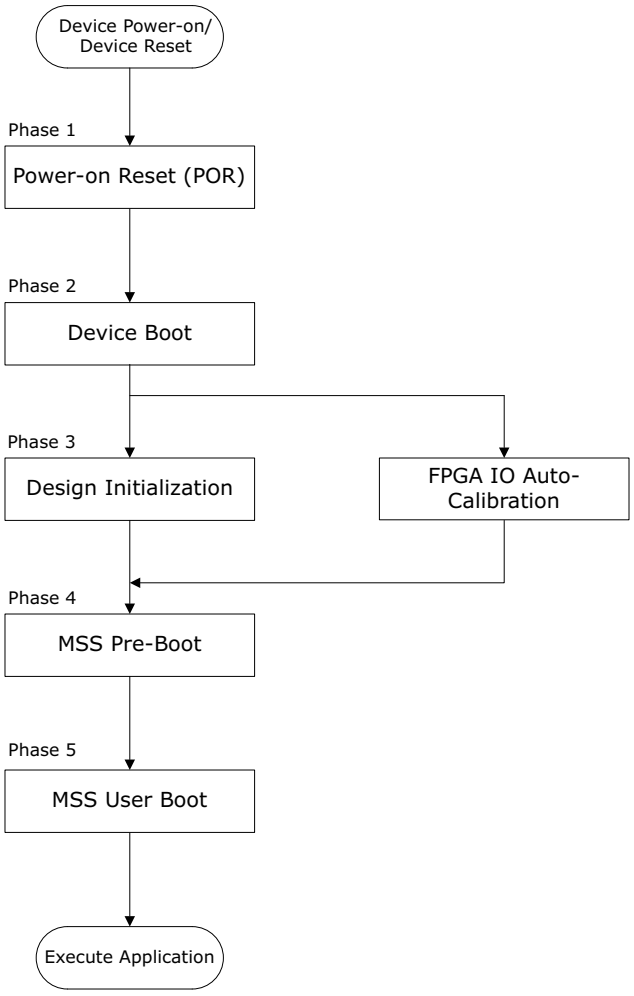
1.1 Boot-up Sequence

The boot-up sequence starts when the PolarFire SoC FPGA is powered-up or reset. It ends when the processor is ready to execute an application program. This booting sequence runs through several stages before it begins the execution of programs.

A set of operations are performed during the Boot-up process that includes power-on reset of the hardware, peripheral initialization, memory initialization, and loading the user-defined application from non-volatile memory to the volatile memory for execution.

The following figure shows different phases of the Boot-up sequence.

Figure 1 • **Boot-up Sequence**



1.1.1 MSS Pre-Boot

Upon successful completion of Design Initialization, MSS Pre-boot starts its execution. The MSS is released from a reset after completion of all normal startup procedures. The system controller manages the programming, initialization, and configuration of the devices. MSS Pre-boot does not occur if the programmed device is configured for system controller suspend mode.

The MSS pre-boot phase of initialization is coordinated by system controller firmware, although it may make use of the E51 in the MSS Core Complex to perform certain parts of the pre-boot sequence.

The following events occur during the MSS pre-boot stage:

- Power-up of the MSS embedded Non-Volatile Memory (eNVM)
- Initialization of the redundancy repair associated with the MSS Core Complex L2 cache
- Authentication of User boot code (if User Secure boot option is enabled)
- Handover operational MSS to User Boot code

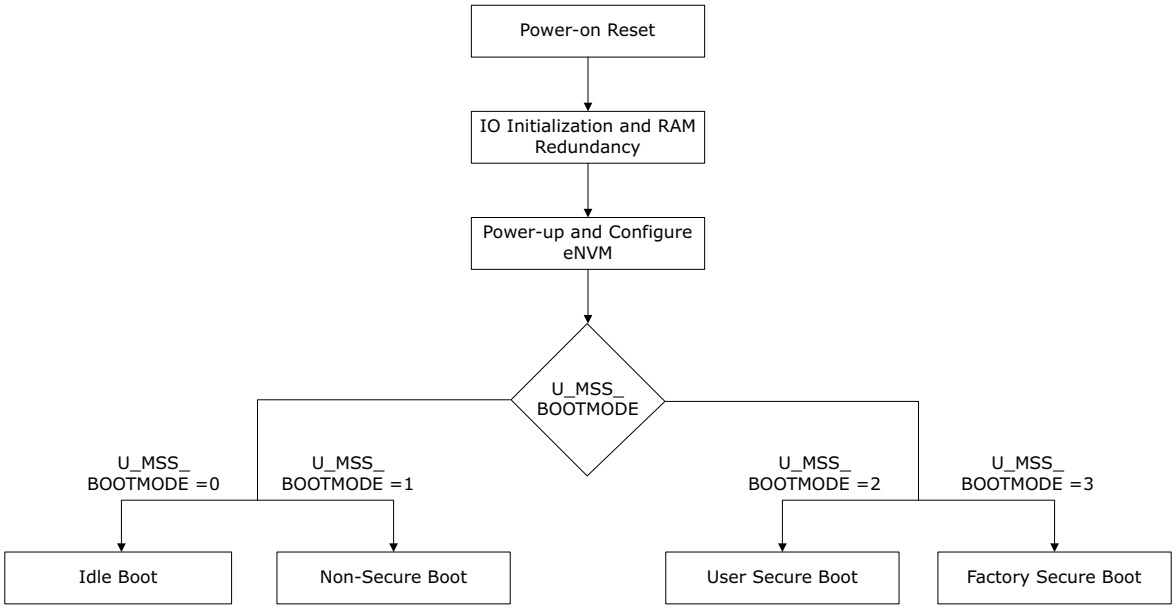
The MSS Core Complex can be booted in one of four modes. The following table lists the MSS pre-boot options, which can be configured and programmed into the sNVM. The boot mode is defined by the user parameter U_MSS_BOOTMODE[1:0]. Additional boot configuration data is mode-dependent and is defined by the user parameter U_MSS_BOOTCFG (see Table 3, page 4 and Table 5, page 6).

Table 1 • **MSS Core Complex Boot Modes**

U_MSS_BOOTMODE[1:0]	Mode	Description
0	Idle boot	MSS Core Complex boots from boot ROM if MSS is not configured
1	Non-secure boot	MSS Core Complex boots directly from address defined by the U_MSS_BOOTADDR
2	User secure boot	MSS Core Complex boots from sNVM
3	Factory secure boot	MSS Core Complex boots using the factory secure boot protocol

The boot option is selected as part of the Libero design flow. Changing the mode can only be achieved through the generation of a new FPGA programming file.

Figure 2 • **MSS Pre-boot Flow**



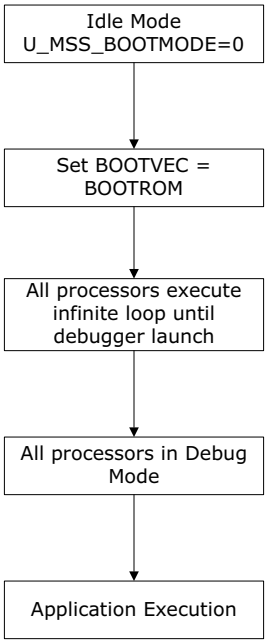
1.1.1.1 Idle Boot

If the MSS is not configured (for example, blank device), then the MSS Core Complex executes a boot ROM program which holds all the processors in an infinite loop until a debugger connects to the target. The boot vector registers maintains their value until the device is reset or a new boot mode configuration is programmed. For configured devices, this mode can be implemented using the U_MSS_BOOTMODE=0 boot option in the Libero configurator.

Note: In this mode, U_MSS_BOOTCFG is not used.

The following figure shows the Idle boot flow.

Figure 3 • **Idle Boot Flow**



1.1.1.2 Non-secure Boot

In this mode, the MSS Core Complex executes from a specified eNVM address without authentication. It provides the fastest boot option, but there is no authentication of the code image. The address can be specified by setting U_MSS_BOOTADDR in the Libero Configurator. This mode can also be used to boot from any FPGA Fabric memory resource through FIC. This mode is implemented using the U_MSS_BOOTMODE=1 boot option.

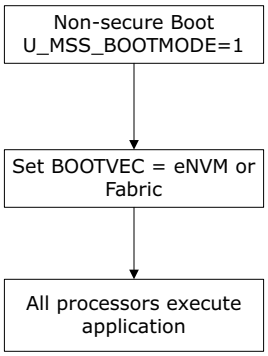
The MSS Core Complex is released from reset with boot vectors defined by U_MSS_BOOTCFG (as listed in the following table).

Table 2 • **U_MSS_BOOTCFG Usage in Non-Secure Boot Mode 1**

Offset (bytes)	Size (bytes)	Name	Description
0	4	$BOOTVEC_0$	Boot vector for E51
4	4	$BOOTVEC_1$	Boot vector for $U54_0$
8	4	$BOOTVEC_2$	Boot vector for $U54_1$
16	4	$BOOTVEC_3$	Boot vector for $U54_2$
20	4	$BOOTVEC_4$	Boot vector for $U54_3$

The following figure shows the Non-secure boot flow.

Figure 4 • **Non-secure Boot Flow**



1.1.1.3 User Secure Boot

This mode allows user to implement their own custom secure boot and the user secure boot code is placed in the sNVM. The sNVM is a 56 KB non-volatile memory that can be protected by the built-in Physically Unclonable Function (PUF). This boot method is considered secured because sNVM pages marked as ROM are immutable. On power up, the system controller copies the user secure boot code from sNVM to Data Tightly Integrated Memory (DTIM) of the E51 Monitor core. E51 starts executing the user secure boot code.

If the size of the user secure boot code is more than the size of the DTIM then user needs to split the boot code into two stages. The sNVM may contain the next stage of the user boot sequence, which may perform authentication of the next boot stage using the user authentication/decryption algorithm.

If authenticated or encrypted pages are used then the same USK key (that is, U_MSS_BOOT_SNVM_USK) must be used for all authenticated/encrypted pages.

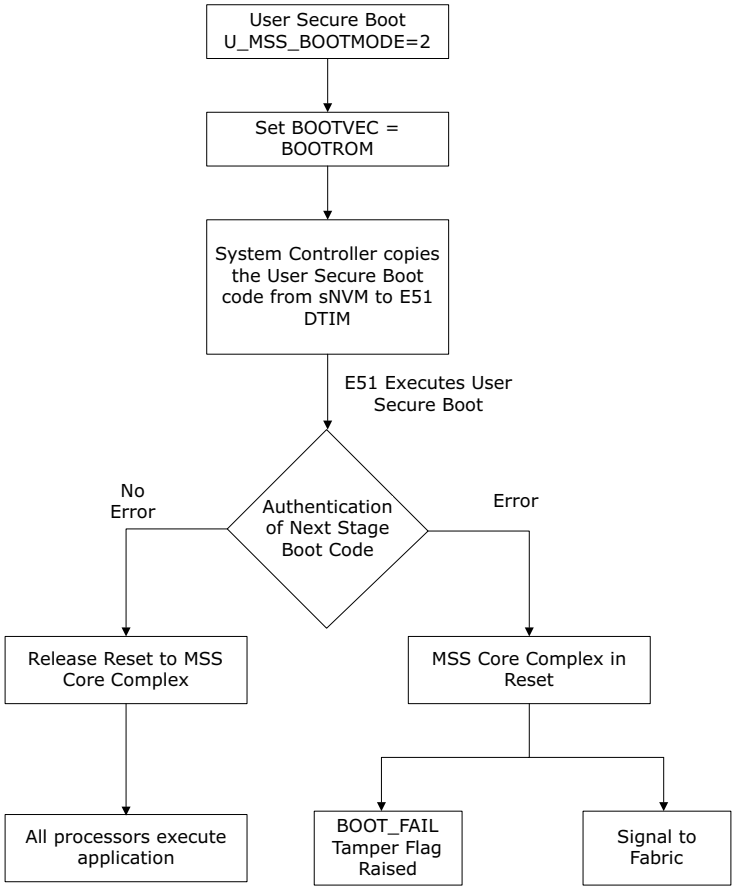
If authentication fails, the MSS Core Complex can be placed in reset and the BOOT_FAIL tamper flag can be raised. This mode is implemented using the U_MSS_BOOTMODE=2 boot option.

Table 3 • **U_MSS_BOOTCFG Usage in User Secure Boot**

Offset (bytes)	Size (bytes)	Name	Description
0	1	U_MSS_BOOT_SNVM_PAGE	Start page in SNVM
1	3	RESERVED	For alignment
4	12	U_MSS_BOOT_SNVM_USK	For authenticated/encrypted pages

The following figure shows the user secure boot flow.

Figure 5 • **User Secure Boot Flow**



1.1.1.4 Factory Secure Boot

In this mode, the system controller reads the Secure Boot Image Certificate (SBIC) from eNVM and validates the SBIC. On successful validation, System Controller copies the factory secure boot code from its private, secure memory area and loads it into the DTIM of the E51 Monitor core. The default secure boot performs a signature check on the eNVM image using SBIC which is stored in eNVM. If no errors are reported, reset is released to the MSS Core Complex. If errors are reported, the MSS Core Complex is placed in reset and the BOOT_FAIL tamper flag is raised. Then, the system controller activates a tamper flag which asserts a signal to the FPGA fabric for user action. This mode is implemented using the U_MSS_BOOTMODE=3 boot option.

The SBIC contains the address, size, hash, and Elliptic Curve Digital Signature Algorithm (ECDSA) signature of the protected binary blob. ECDSA offers a variant of the Digital Signature Algorithm which uses elliptic curve cryptography. It also contains the reset vector for each Hardware thread/core/processor core (Hart) in the system.

Table 4 • **Secure Boot Image Certificate (SBIC)**

Offset	Size (bytes)	Value	Description
0	4	IMAGEADDR	Address of UBL in MSS memory map
4	4	IMAGELEN	Size of UBL in bytes
8	4	$BOOTVEC_0$	Boot vector in UBL for E51
12	4	$BOOTVEC_1$	Boot vector in UBL for U540
16	4	$BOOTVEC_2$	Boot vector in UBL for U541
20	4	$BOOTVEC_3$	Boot vector in UBL for U542
24	4	$BOOTVEC_4$	Boot vector in UBL for U543
28	1	OPTIONS[7:0]	SBIC options
28	3	RESERVED	
32	8	VERSION	SBIC/Image version
40	16	DSN	Optional DSN binding
56	48	H	UBL image SHA-384 hash
104	104	CODESIG	DER-encoded ECDSA signature
Total	208	Bytes	

DSN

If the DSN field is non-zero, it is compared against the device's own serial number. If the comparison fails, then the boot_fail tamper flag is set and authentication is aborted.

VERSION

If SBIC revocation is enabled by U_MSS_REVOCATION_ENABLE, the SBIC is rejected unless the value of VERSION is greater than or equal to the revocation threshold.

SBIC REVOCATION OPTION

If SBIC revocation is enabled by U_MSS_REVOCATION_ENABLE and OPTIONS[0] is ‘1’, all the SBIC versions less than VERSION are revoked upon complete authentication of the SBIC. The revocation threshold remains at the new value until it increments again by a future SBIC with OPTIONS[0] = ‘1’ and a higher VERSION field. The revocation threshold may only be incremented using this mechanism and can only be reset by a bit-stream.

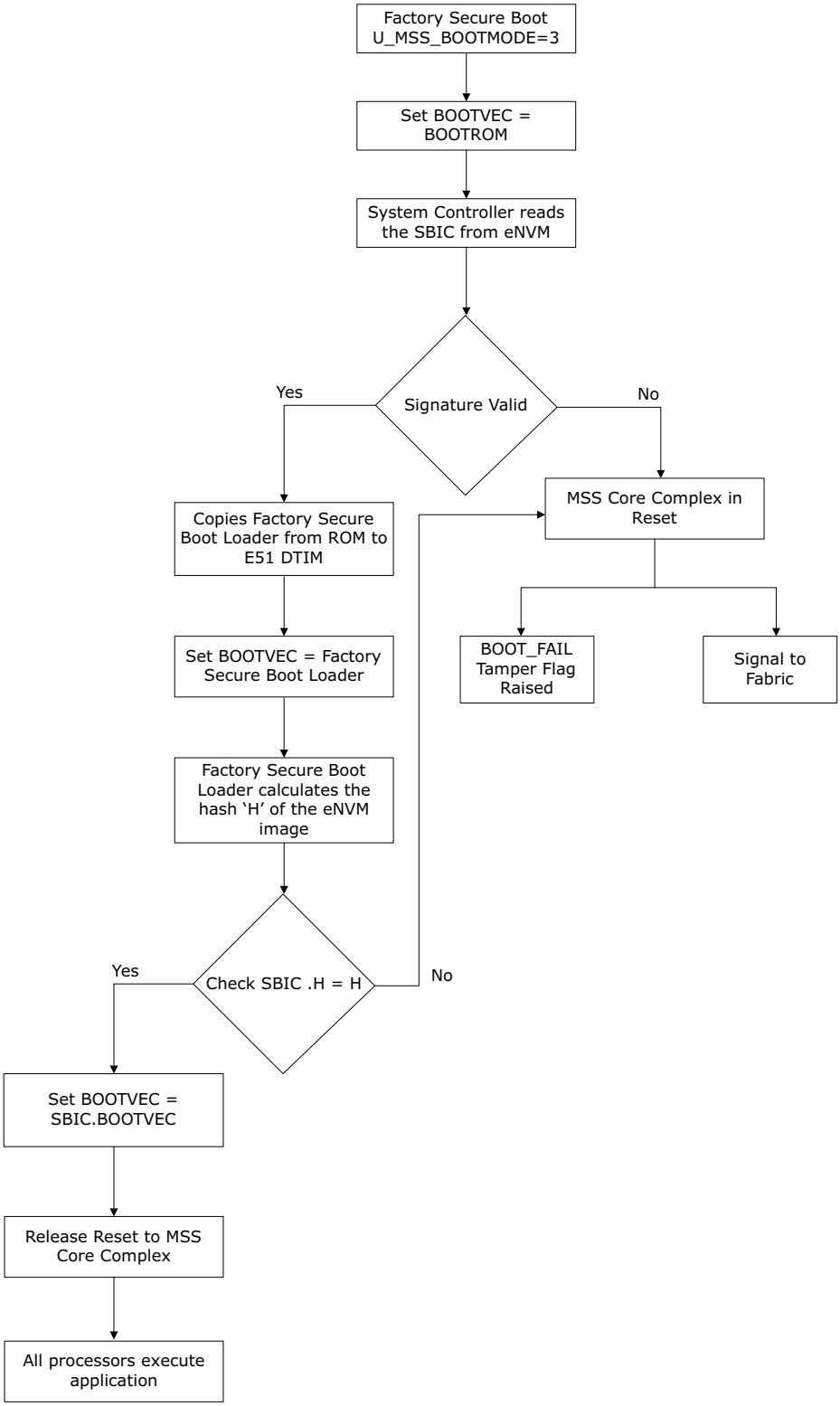
When the revocation threshold is updated dynamically, the threshold is stored using the redundant storage scheme used for passcodes such that a power failure during device boot does not cause a subsequent device boot to fail. If the update of revocation threshold fails, it is guaranteed that the threshold value is either the new value or the previous one.

Table 5 • **U_MSS_BOOTCFG Usage in Factory Boot Loader Mode**

Offset (bytes)	Size (bytes)	Name	Description
0	4	U_MSS_SBIC_ADDR	Address of SBIC in MSS address space
4	4	U_MSS_REVOCATION_ENABLE	Enable SBIC revocation if non-zero

The following figure shows the factory secure boot flow.

Figure 6 • **Factory Secure Boot Flow**



1.1.2 MSS User Boot

MSS user boot takes place when the control is given from System Controller to MSS Core Complex. Upon successful MSS pre-boot, system controller releases the reset to the MSS Core Complex. MSS can be booted up in one of the following ways:

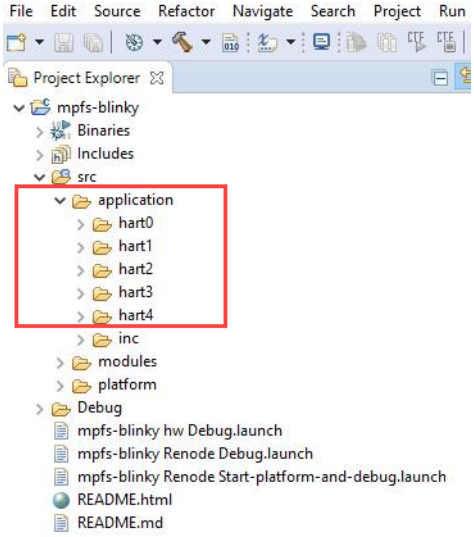
- Bare Metal Application
- Linux Application
- AMP Application

1.1.2.1 Bare Metal Application

The bare metal applications for the PolarFire SoC can be developed using SoftConsole tool. This tool provides the output files in the form of `.hex` which can be used in the Libero flow to include into the programming bitstream file. The same tool can be used to debug the Bare Metal applications using JTAG interface.

Th following figure shows the SoftConsole Bare Metal application which has five harts (Cores) including E51 Monitor core.

Figure 7 • **SoftConsole Project**



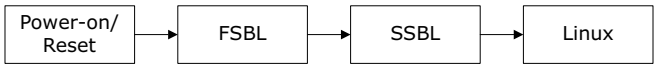
1.1.2.2 Linux Application

This section describes the boot sequence for Linux running on all U54 cores.

A typical boot process consists of three stages. The first stage boot loader (FSBL) gets executed from the on-chip Boot flash (eNVM). The FSBL loads the second stage boot loader (SSBL) from a boot device to external RAM or Cache. The boot device can be eNVM or embedded memory microcontroller (eMMC) or external SPI Flash. The SSBL loads the Linux operating system from boot device to external RAM. In the third stage, Linux is executed from the external RAM.

The following figure shows the Linux Boot Process flow.

Figure 8 • **Typical Linux Boot Process Flow**



Details of FSBL, Device tree, Linux, and YOCTO build, how to build and configure Linux will be provided in the future release of this document.

1.1.2.3 AMP Application

Detailed description of Libero MSS Configurator and how to debug multi-processor applications using SoftConsole will be provided in the future release of this document.

1.2 Different Sources of Booting

To be updated in future versions of this document.

1.3 Boot Configuration

To be updated in future versions of this document.

2 Acronyms

The following acronyms are used in this document.

Table 1 • **List of Acronyms**

Acronym	Expanded
AMP	Asymmetric Multi-processing
DTIM	Data Tightly Integrated Memory (also called as SRAM)
ECDSA	Elliptic Curve Digital Signature Algorithm
eNVM	embedded Non-Volatile Memory
FSBL	First Stage Boot Loader
Hart	Hardware thread/core/processor core
MSS	Microprocessor Subsystem
POR	Power on Reset
PUF	Physically Unclonable Function
ROM	Read-only Memory
SCB	System Controller Bridge
sNVM	Secure Non-volatile Memory

3 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

3.1 Revision 2.0

The following is a summary of the changes made in this revision.

- Information about Factory Secure Boot was updated.
- Information about Bare Metal Application was updated.

3.2 Revision 1.0

The first publication of this document.