



AMD Platform Security Processor BIOS Architecture Design Guide for AMD Family 17h and Family 19h Processors

Publication # 55758	Revision: 1.13
Issue Date: June 2021	

© 2014 – 2021 Advanced Micro Devices, Inc. All rights reserved.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. (“AMD”) products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD’s Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD’s products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD’s product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Trademarks

AMD, the AMD Arrow logo, AGESA, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Reverse engineering or disassembly is prohibited.

ARM and TrustZone are registered trademarks of ARM Limited.

Microsoft and Windows, are registered trademarks of Microsoft Corporation.

PCIe is a registered trademark of PCI-Special Interest Group (PCI-SIG).

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG ACTUAL OR DE FACTO VIDEO AND/OR AUDIO STANDARDS IS EXPRESSLY PROHIBITED WITHOUT ALL NECESSARY LICENSES UNDER APPLICABLE PATENTS. SUCH LICENSES MAY BE ACQUIRED FROM VARIOUS THIRD PARTIES INCLUDING, BUT NOT LIMITED TO, IN THE MPEG PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

Contents

List of Figures	7
List of Tables	9
Revision History	11
Definitions	17
References	20
Chapter 1 Introduction	21
1.1 Scope	21
1.1.1 PSP Overview	21
1.1.2 Key Features of the PSP	21
Chapter 2 Overview of Feature Implementation	23
2.1 Platform Secure Boot	23
2.2 Firmware TPM Functions (Client PSP Only)	24
Chapter 3 PSP Components	25
3.1 On-chip PSP Boot ROM	25
3.2 Off-chip PSP Boot Loader	25
3.2.1 AMD Family 17h Models 00h–0Fh Processor PSP Boot Loader	25
3.3 Off-chip PSP Secure OS for Family 17h	31
3.4 PSP AGESA™ Binaries	32
Chapter 4 Overview of BIOS Support for PSP	33
4.1 BIOS Build Process	33
4.1.1 Build SPI Image	33
4.1.2 BIOS Build Flow	33
4.1.3 Directory Table	36
4.1.4 PSP Directory Table	42
4.1.5 BIOS Directory Table (AMD Family 17h and 19h Processor)	54
4.1.6 SubProgram Field Within a Chip Family	59
4.1.7 EFS Search Algorithm	60
4.2 Runtime Execution Flow	70
4.2.1 BIOS Boot x86 Initialization	70

4.2.2	BIOS Runtime Functionality	72
4.3	Optimized Boot Flow	72
4.4	APCB Recovery	74
4.5	PSP Initiated Crisis A/B Recovery Path	76
4.5.1	Overview	76
4.5.2	PSP/BIOS Directory Update	76
4.5.3	Recovery Scenario for PSP Boot Loader	79
4.5.4	Platform Secure Boot Related Changes	80
4.5.5	SBIOS Implementation Details	80
4.5.6	A/B Recovery Changes for Family 19h Models 40h-4fh Onward	80
4.6	PSP/BIOS Directory Upgrade Progress	83
4.7	Firmware Anti-rollback BIOS Requirement	83
4.8	How to Add RPMC Support in BIOS	85
4.9	Use AmdPspPsbFusingLib to Customize PSB Fusing	85
Chapter 5	X86 BIOS S3-Resume Path Handling	87
5.1	BIOS S3 Transition Flow on ACPI Aware OS	87
5.2	BIOS S3 Resume	87
5.2.1	Modified Conventional Resume	87
Chapter 6	TPM Software Interface	89
6.1	TPM 2.0 Command/Response Buffer Interface	89
6.2	AMD Implementation of TPM 2.0 Interface	90
6.2.1	Disable fTPM	91
6.2.2	Clear fTPM NVRAM	91
Chapter 7	BIOS PSP Mailbox Interaction	93
7.1	BIOS to PSP Mailbox	93
7.1.1	BIOS to PSP Mailbox Commands	96
7.2	PSP-to-BIOS Mailbox	104
7.2.1	PSP to BIOS Mailbox Commands	104
7.3	MP0_P2C_MSG reserved for X86 and PSP information sharing	108
Chapter 8	Platform BIOS Requirements for PSP Implementation	109
Chapter 9	AMD AGESA™ PI Interface Note	113

9.1	Library	113
9.2	Drivers	113
9.3	UEFI PPI/Protocol Consumed/Produced by PSP Drivers	115
9.3.1	PSP_FTPM_PPI	115
9.3.2	PSP fTPM Protocol	116
9.3.3	AMD_PSP_PLATFORM_PROTOCOL	117
9.3.4	AMD_PSP_RESUME_SERVICE_PROTOCOL	119
Chapter 10	Standards	121
10.1	UEFI 2.3.1c Chapter 27 Secure Boot	121
10.2	Microsoft® Trusted Execution Environment UEFI Protocol	121
10.3	Microsoft® Trusted Execution Environment ACPI Profile	121
10.4	AMD PSP 1.0 Software Architecture Design Document	121
Appendix A	PSP S5 Boot Flow	123
A.1	Boot Flow — S5 Cold Boot	123
Appendix B	BuildPspDirectory Tool Version 4.x	125
B.1	PSP Directory Configure File Format	125
B.1.1	Node <DIRS>	125
B.1.2	Node <PSP_DIR>	127
B.1.3	Node <BIOS_DIR>	131
B.1.4	Node <COMBO_DIR>	133
B.1.5	ISH Header	134
B.2	Command Line Parameters	136
B.2.1	Positional Arguments	136
B.2.2	Optional Arguments	136
B.2.3	Build Directory Table	136
Appendix C	Modified Conventional Resume S3 (SMM->SEC->PEI) Design Guideline ...	139
C.1	Introduction	139
C.2	Design Discussion	140
C.3	Platform BIOS Porting Details	142
Appendix D	Postcode Definition for PSP FW	145
Appendix E	HSTI Bitmap Definition	147

E.1	Security Feature Byte Index 0	147
E.2	Security Feature Byte Index 1	147
E.3	Security Feature Byte Index 2	147
E.4	Security Feature Byte Index 3	148
E.5	Security Feature Byte Index 4	148
E.6	Security Feature Byte Index 5	148
Appendix F	FIPS Certification on AMD FP6 Platform	149

List of Figures

Figure 1.	Platform Secure Boot Overview	23
Figure 2.	BIOS Build Flow Summary Diagram.	34
Figure 3.	PSP Directory Table	42
Figure 4.	PSP Directory, With or Without Combo Directory	53
Figure 5.	APCB Recovery Flow	75
Figure 6.	Layout of BIOS Image with 2 Level Directories	77
Figure 7.	PSB Fusing Process Flow	86
Figure 8.	TPM2 Command/Response Buffer Interface	90
Figure 9.	BIOS-PSP Mailbox Interface	93
Figure 10.	BIOS-PSP Mailbox Command Execution Sequence	94
Figure 11.	Enable PSB Fusing Command Processing	103
Figure 12.	AMD Family 17h Models 00h–0Fh Processor Boot Flow — S5 Cold Boot	124
Figure 13.	Cold Boot Flow.	141
Figure 14.	FIPS Mode Disabled.	150
Figure 15.	FIPS Mode Enabled, Self-Tests Passed	150

List of Tables

Table 1.	Definitions, Acronyms and Abbreviations	17
Table 2.	Embedded Firmware Structure	37
Table 3.	PSP Directory Table Header Fields	42
Table 4.	PSP Directory Table Additional Info Fields	43
Table 5.	PSP Directory Table Entry Fields	43
Table 6.	PSP Entry Bit Field Definition	43
Table 7.	Location Bit Field Definition	44
Table 8.	AMD Family 17h and 19h Processor PSP Directory Type Encodings.	44
Table 9.	PSP Soft Fuse Chain Definition	49
Table 10.	PSP/BIOS Combo Directory Header	51
Table 11.	PSP Combo Directory Entry Fields	52
Table 12.	Valid PSP IDs per Program.	53
Table 13.	Legacy BIOS in PSP L1 Minimum Entries	54
Table 14.	A/B BIOS in PSP L1 Minimum Entries	54
Table 15.	BIOS Directory Table Header Fields	55
Table 16.	BIOS Directory Table Entry Fields	55
Table 17.	BIOS Directory Table Entries	57
Table 18.	PMU Firmware Subtype Encoding	58
Table 19.	Standard BIOS Binary Header	59
Table 20.	Sub-Programming Encoding	59
Table 21.	Fuse Bits to Assist EFS Search Algorithm	60
Table 22.	Fuse Bits to Assist EFS Search Algorithm	63
Table 23.	Fuse Bits to Assist EFS Search Algorithm	67
Table 24.	PSP Directory Level 1	77
Table 25.	PSP Directory Level 2	78
Table 26.	BIOS Directory Level 2	79
Table 27.	ISH Structure Definition	81
Table 28.	Corruption Reporting Log Format	83

Table 29.	Control Area Layout	89
Table 30.	BIOS-PSP Mailbox Status Register Bit Fields	94
Table 31.	BIOS-to-PSP Mailbox Commands	95
Table 32.	Bit Definitions for Capability Field	99
Table 33.	PSP-to-BIOS Mailbox Commands	104

Revision History

Date	Revision	Description
June 2021	1.13	<p>Updated version numbers in “References” on page 20.</p> <p>Chapter 4 updates:</p> <ul style="list-style-type: none"> Updated offsets for Family 19h in Section 4.1.3, “Directory Table” on page 36. Updated Max Size in Table 4, “PSP Directory Table Additional Info Fields” on page 43 Added/updated: <ul style="list-style-type: none"> Table 5, “PSP Directory Table Entry Fields” on page 43 Table 6, “PSP Entry Bit Field Definition” on page 43 Table 7, “Location Bit Field Definition” on page 44 Updated Table 8, “AMD Family 17h and 19h Processor PSP Directory Type Encodings” on page 44. Clarified Cookie description in Table 10, “PSP/BIOS Combo Directory Header” on page 51. Updated Table 12, “Valid PSP IDs per Program” on page 53. Updated Table 16, “BIOS Directory Table Entry Fields” on page 55. Updated Table 17, “BIOS Directory Table Entries” on page 57. Renamed and updated Section 4.1.7, “EFS Search Algorithm” on page 60 for AMD Family 17h and AMD Family 19h processors. Updated Section 4.7, “Firmware Anti-rollback BIOS Requirement” on page 83. <p>Appendix B updates:</p> <ul style="list-style-type: none"> Updated Appendix B, “BuildPspDirectory Tool Version 4.x” on page 125 introductory text. Added values and examples in B.1.1, “Node <DIRS>” on page 125. Updated File, Size, and AddressMode attributes in B.1.2, “Node <PSP_DIR>” on page 127. <p>Appendix D updates:</p> <ul style="list-style-type: none"> Updated Appendix D, “Postcode Definition for PSP FW” on page 145 introductory text. <p>Appendix F updates:</p> <ul style="list-style-type: none"> Updated Appendix F, “FIPS Certification on AMD FP6 Platform” on page 149 introductory text.

Date	Revision	Description
March 2021	1.12	<p>Chapter 2 updates:</p> <ul style="list-style-type: none"> Added implementation references to Section 2.1, “Platform Secure Boot” on page 23. <p>Chapter 3 updates:</p> <ul style="list-style-type: none"> Updated Section 3.1, “On-chip PSP Boot ROM” on page 25 for firmware anti-rollback. Updated Section 3.2.1, “AMD Family 17h Models 00h–0Fh Processor PSP Boot Loader” on page 25, for sequence, added steps, and SP5 socket. <p>Chapter 4 updates:</p> <ul style="list-style-type: none"> Updated Section 4.1.3, “Directory Table” on page 36 recommendations to account for Multi Gen EFS. Updated Table 2, “Embedded Firmware Structure” on page 37. Updated Table 5, “PSP Directory Table Entry Fields” on page 43. Updated Table 8, “AMD Family 17h and 19h Processor PSP Directory Type Encodings” on page 44. Updated and added rows to Table 9, “PSP Soft Fuse Chain Definition” on page 49. Updated Table 12, “Valid PSP IDs per Program” on page 53. Added Section 4.1.4.3, “One Level PSP Directory Layout” on page 54. Updated Table 20, “Sub-Programming Encoding” on page 59. Updated steps, examples in Section 4.1.8, “Server Product Build Changes Starting with Family 19h Model 00h-0Fh” on page 62. Updated Table 19, “Multi Gen EFS Values” on page 63 Added Section 4.5.6, “A/B Recovery Changes for Family 19h Models 40h-4fh Onward” on page 80. Updated Section 4.7, “Firmware Anti-rollback BIOS Requirement” on page 83. <p>Chapter 6 updates:</p> <ul style="list-style-type: none"> Updated Figure 8 on page 90. <p>Chapter 7 updates:</p> <ul style="list-style-type: none"> Removed “MboxBiosCmdClrSmmLock (MboxCmd = 0x17)” from Table 31 and Section 7.1.1, “BIOS to PSP Mailbox Commands”. <p>Appendix updates:</p> <ul style="list-style-type: none"> Added B.1.5, “ISH Header” on page 134. Removed Appendix, “Key Format.” Removed Appendix, “Enabling PSP-based OEM System Trusted Application.” Added Appendix F, “FIPS Certification on AMD FP6 Platform” on page 149.

Date	Revision	Description
August 2020	1.11	<ul style="list-style-type: none"> Added AMD Family 19h Model 20h-2Fh Processors. Updated Table 2 on page 37. Updated Table 3 on page 42. Added Table 4 on page 43. Updated Table 5 on page 43. Updated and added rows to Table 8 on page 44. Added rows to Table 9 on page 49. Updated Table 15 on page 55. Updated and added rows to Table 16 on page 55. Added rows to Table 17 on page 57. Updated Section 4.1.6, “SubProgram Field Within a Chip Family” on page 59. Added sub-program rows to Table 20 on page 59. Updated Section 4.1.7 title: “Client Product Build Changes Starting with Family 17h Models 30h-3Fh” on page 60. Added Section 4.1.8: “Server Product Build Changes Starting with Family 19h Model 00h-0Fh” on page 62. Updated Section 4.5.1, “Overview” on page 76 for A/B recovery. Added row to Table 24 on page 77. Added Section 4.7, “Firmware Anti-rollback BIOS Requirement” on page 83. Added Section 4.8, “How to Add RPMC Support in BIOS” on page 85. Added Section 4.9, “Use AmdPspPsbFusingLib to Customize PSB Fusing” on page 85. Added rows to Table 31 on page 95. Updated Section 7.2, “PSP-to-BIOS Mailbox” on page 104. Updated Appendix B.1.2 “Node <PSP_DIR>” on page 127.
March 2020	1.10	<ul style="list-style-type: none"> Add Family 19h to the title. Updated title to Section 3.2.1, AMD Family 17h and 19h Processor PSP Boot Loader. Updated Family and Model information items 49 and 50 in Section 3.2.1, AMD Family 17h and 19h Processor PSP Boot Loader.

Date	Revision	Description
February 2020	1.09	<ul style="list-style-type: none"> Added AMD Family 19h Models 20h–2Fh Updated Section 4, Overview of BIOS Support <ul style="list-style-type: none"> Table 2. Embedded Structure Table 5. AMD Family 17h Processor PSP Directory Type Encodings Table 6. PSP Soft Fuse Chain Definition Table 9. Valid PSP IDs per Program Updated Table 5. AMD Family 17h Processor PSP Directory Type Encodings. Updated Section 6.2.2, Clear fTPM NVRAM. Updated Section 7, BIOS PSP Mailbox Interaction: <ul style="list-style-type: none"> Section 7.1.1.2, MboxBiosCmdLockSpi (MboxCmd = 0x1F) Section 7.1.1.3, MboxBiosCmdPspQuery (MboxCmd = 0x05). Removed Appendix A, PSP Directory Structure Removed Appendix B, BIOS Directory Structure Updated Appendix C, BuildPspDirectory Tool Version 4: <ul style="list-style-type: none"> Section C.1.2, Node <PSP_DIR>
February 2020	1.08	<ul style="list-style-type: none"> Special NDA release that was not posted on DevHub.
February 2020	1.08	<ul style="list-style-type: none"> Added Family 19h Models 00h–0Fh Updated Section 4 Overview of BIOS Support <ul style="list-style-type: none"> Table 2. Embedded Structure Table 6. PSP Soft Fuse Chain Definition Table 9. Valid PSP IDs per Program Updated Table 5. AMD Family 17h Processor PSP Directory Type Encodings. Updated Section 6.2.2 Clear fTPM NVRAM. Updated Section 7 BIOS PSP Mailbox Interaction: <ul style="list-style-type: none"> Section 7.1.1.2 MboxBiosCmdLockSpi (MboxCmd = 0x1F) Section 7.1.1.3 MboxBiosCmdPspQuery (MboxCmd = 0x05). Removed Appendix A PSP Directory Structure Removed Appendix B BIOS Directory Structure Updated Appendix C BuildPspDirectory Tool Version 4: <ul style="list-style-type: none"> Section C.1.2 Node <PSP_DIR>

Date	Revision	Description
November 2018	1.07	<ul style="list-style-type: none"> • Update 4.13 Directory Table, use offset instead physical MMIO address. • Update Table 2. Embedded Firmware Structure, with new bit. second_gen_efs, and new SPI setting. • Add new PSP firmware entry type. • Add new section 4.1.7 "Build changes start from Family 17h Models 30h-3Fh", which address the build changes to support 32M BIOS image. • Add section to describe "RomId" which used to support two SPI user case. • Add new chipid for Family 17h Models 70h-7Fh in Combo directory. • Update new commands in section 7.1 BIOS to PSP Mailbox. • Add new attribute "HeaderBase" in Appendix E BuildPspDirectory Tool Version 3.x. • Add section Node <COMBO_DIR> in Appendix E BuildPspDirectory Tool Version 3.x.
May 2018	1.06	<ul style="list-style-type: none"> • Update 4.13 Directory Table, to add the address for 32M SPI, entry in Embedded Firmware Structure for BIOS Directory table for Family 17h Models 30h-3Fh • Update Table 4 PSP Directory Table Entry Fields to add new field "SubProgram" • Update Table 5 AMD Family 17h Processor PSP Directory Type Encodings, to add new PSP Directory types • Add Table 6 PSP Soft Fuse Chain definition • Update Table 8 Valid PSP IDs per Program, to add new fields for Family 17h Models 30h-3Fh • Update Table 11 BIOS Directory Table Entry Fields, to add new field "SubProgram" • Update Table 20. BIOS-to-PSP Mailbox Commands, to add new C2P command
December 2017	1.05	<ul style="list-style-type: none"> • Updated the document to state Platform Secure Boot. • Updated Section 4.1.2 BIOS Build Flow and Section 4.1.3 Directory Table. • Updated Figure 3. • Updated Table 5, Table 9, Table 10, Table 11. • Updated Section 4.2.1 BIOS Boot x86 Initialization. • Updated Section 4.5.1 Overview. • Updated Section 5.2.1 Modified Conventional Resume. • Updated Appendix B BIOS Directory Structure. • Updated Section E.2.2 Build Directory Table.
May 2017	1.04	<ul style="list-style-type: none"> • Updated Table 6. Directory Type Encodings . • Updated Table 13 BIOS Directory Entries. • Updated Table 17 PSP Directory Level 2. • Modified Section 6.2.2 Swapping Processor when fTPM Enabled. • Updated Table 21 BIOS-to-PSP Mailbox Commands. • Modified Section H4. Sending command to OEM TA from BIOS.

Date	Revision	Description
December 2016	1.03	<ul style="list-style-type: none"> • Updated Embedded Firmware Structure • Updated AMD Family 17h Processor PSP Directory Type Encodings • Updated the Reserve CMOS shadow region address for APCB recovery • Added PSP initiated Crisis Recovery Path section • Added PSP/BIOS Directory Upgrade Progress section • Added Disable fTPM section • Added Swapping Processor when fTPM Enabled section • Updated PSP FW FW_STATUS • Added HSTI Bitmap Definition section
September 2016	1.02	<p>Added new statement into the Introduction/Scope.</p> <p>Updated Integrated Trusted Platform Module (TPM) Functions.</p> <p>Included SP4 into AMD Family 17h Models 00h–0Fh Processor PSP Boot Loader section.</p> <p>Removed SOi3 from document.</p> <p>Updated Directory Table information.</p> <p>Updated Table 6, Table 12 and Table 13.</p> <p>Added new Section 4.4, APCB Recovery.</p> <p>Updated Figure 8.</p> <p>Removed Figure 11 and Figure 12, replaced with new Figure 11.</p> <p>Modified Appendix F, Section F3, Platform BIOS Porting Details.</p> <p>Added Force AP to halt in Security phase in Appendix F</p>
July 2016	1.01	<p>Added 0x42 through 0x45 Offset in Table 3.</p> <p>Changed Description of PSP Cookie in Table 8.</p> <p>Changed Offset and Size for Reserved field in Table 8</p> <p>Modified Table 10 to include AMD Family 17h Models 10h–1Fh.</p> <p>Added new row for 0x66 offset in Table 13.</p> <p>Added new row for 0x05 offset in Table 14.</p> <p>Modified the Note in Section 3.4 PSP AGESA™ Binaries</p> <p>Corrected the sentence in Section 4.1.2.1 concerning size of signature data.</p> <p>Changed Section C.0.1 to Section C.1.</p> <p>Changed Section 4.1.3.2 to 4.1.3. All subsequent sub heading have been changed accordingly.</p> <p>Changed Section 4.1.3.3 to 4.1.4. All subsequent sub heading have been changed accordingly.</p> <p>Changed Section 4.1.3.4 to 4.1.5. All subsequent sub heading have been changed accordingly.</p>
March 2016	1.00	Initial Release.

Definitions

Table 1. Definitions, Acronyms and Abbreviations

Term	Definition	Comments
AES	Advanced Encryption Standard	
AGESA™	AMD Generic Encapsulated Software Architecture	AMD package that includes the firmware to initialize Silicon
AP	Application Processor	Secondary core in a multi-core cluster
CCP	Cryptographic Co-Processor	
CRTM	Core Root of Trust for Measurement	
DMA	Direct Memory Access	
DRAM	Dynamic Random Access memory	
DXE	Driver Execution Environment	Driver Execution environment phase, that run after memory has been initialized.
ECC	Elliptic Curve Cryptography	
EFI	Extensible Firmware Interface	
FFS	Firmware File system	A binary storage form that is well suited to firmware volumes. The abstracted model of the FFS is a flat file system
fTPM	Firmware TPM	Firmware emulated TPM
FV	Firmware Volume	A FV is a simple Flash File System that starts with a header and contains files that are named by a GUID. The file system is flat and does not support directories. Each file is made up of a series of sections that support encapsulation.
FW	Firmware	
HOB	Hand-Off Block	A structure used to pass information from one boot phase to another (i.e., from the PEI phase to the DXE phase)
HMAC	Keyed-Hash Message Authentication Code	In cryptography, a keyed-hash message authentication code (HMAC) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret cryptographic key.

Table 1. Definitions, Acronyms and Abbreviations(Continued)

Term	Definition	Comments
HSM	Hardware Security Module	A hardware security module (HSM) is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing without revealing keying material
IBV	Independent BIOS Vendor	
MTM	Mobile Trusted Module	A firmware version of a TPM
OEM	Original Equipment Manufacturer	
OS	Operating System	
PEI	Pre-EFI Initialization	Set of drivers usually designed to initialize memory and the CPU so that DXE phase can run.
PKCS	Public Key Cryptography Standards	
PSP	Platform Security Processor	
RNG	Random Number Generator	
ROM	Read Only Memory	
RoT	Root of Trust	
RSA	Rivest-Shamire-Adleman encryption algorithm	
RTM	Root of trust for measurement	
SEC	Security Phase	Initial starting point for boot process, first code executed after hardware reset. Responsible for 1) Establishing root trust in the software space; 2) Initializing architecture specific configuration to establish memory space for the C code stack.
SHA	Secure Hash Algorithm	
SMM	System Management Mode	An operating mode where all normal execution (including the operating system) is suspended, and special separate software (usually firmware or a hardware-assisted debugger) is executed in high-privilege mode.
SPI	Serial Peripheral Interface Bus	Also referred to as the Non-volatile ROM chip on this Bus
SRAM	Static Random Access Memory	
TCG	Trusted Computing Group	A standards organization
TEE	Trusted Execution Environment	ARM® TrustZone® is one example of a technology that establishes a TEE
TPM	Trusted Platform Module	A hardware root of trust

Table 1. Definitions, Acronyms and Abbreviations(Continued)

Term	Definition	Comments
	AMD Signing Key	A 2048 bit RSA key pair generated by AMD. The private key is used to sign the public portion of OEM signing key
	OEM Signing Key	An asymmetric key pair generated by OEMs. The private key is used to sign the RTM volume of BIOS. The public portion of signed OEM key is stored in the SPI BIOS image
	BIOS RTM Volume	BIOS firmware Volume that is root of trust of x86 BIOS execution. The code in this volume is executed at x86reset. Based on OEM implementation this can be SEC volume or combined SEC-PEI volume. PSP firmware authenticates BIOS RTM volume before releasing the main core.
	PSP Directory	A simple directory at certain SPI location that lists various firmware images and respective location in the SPI space

References

1. *Processor Programming Reference (PPR) for AMD Family 17h Models 10h–1Fh Processors*, order# 55570.
2. *Unified Extensible Firmware Interface Specification*, Version 2.8 or newer, Errata B.
3. *Unified Extensible Firmware Platform Initialization Specification*, Version 1.7 or newer
4. *Trusted Platform Module Library Specification*, Family "2.0", Level 00, Revision 00.99, 31-Oct-2013.
5. *Microsoft TPM v2.0 Command and Signal Profile*, July 26, 2013.
6. *Trusted Execution Environment EFI Protocol*, version 0.9, December 9, 2011.
7. *TPM Command/Response Buffer Interface w/Locality Support*, Version 0.56, DRAFT, 01-09-2013.
8. *Enabling Platform Secure Boot for AMD Family 17h Processor Based Client Platforms User's Guide*, order# 56654
9. *Enabling Platform Secure Boot for AMD Family 17h Models 00h–0Fh and 30h–3Fh and Family 19h Models 00h–0Fh Processor-Based Server Platforms*, order# 56534.

Chapter 1 Introduction

1.1 Scope

This document refers to the AMD Secure Processor technology as Platform Security Processor (PSP). This document's primary focus is to cover BIOS requirements and to suggest implementation guidelines for AMD Family 17h and 19h. This document does not cover the details of the Platform Security Processor (PSP) firmware or PSP functionality. This document covers only the services and interfaces that BIOS provides to PSP firmware plus the boot flow and boot media layout impact of PSP.

1.1.1 PSP Overview

The Platform Security Processor (PSP) is an isolated security processor that runs independently from the main cores of the platform — where security sensitive components can run without being affected by the commodity, untrusted software running as the main system workload. PSP executes its own firmware and shares the SPI flash storage that is used by BIOS or use a separate SPI-ROM accessible only by PSP, depending on platform design.

1.1.2 Key Features of the PSP

1.1.2.1 Platform Secure Boot

- Platform Secure Boot formerly known as Hardware Validated Boot.
- The PSP validates the signature of the initial BIOS Boot code prior to starting BIOS boot. PSP is the Core Root of Trust for Measurement (CRTM) and the main cores are only released from reset if the BIOS image is authentic.
- Only validated BIOS is allowed to boot.
- The initial block of BIOS code is responsible for subsequently validating the signatures of all other BIOS code blocks loaded from the system read only memory (ROM).

1.1.2.2 Integrated Trusted Platform Module (TPM) Functions

Implements the TPM 2.0 functions required (for some categories of systems) by Microsoft® Windows® 8, Windows 10, and the Windows 10 update, codenamed “Redstone”.

1.1.2.3 Cryptographic Offload Support

Provides hardware acceleration of cryptographic algorithms for PSP FW. Also provides true random number generator (RNG) support accessible using the RDRAND x86 CPU instruction.

Chapter 2 Overview of Feature Implementation

2.1 Platform Secure Boot

Platform Secure Boot is an AMD specific form of secure boot, that roots the trust to hardware in an immutable PSP on-chip ROM firmware and validates the integrity of the system ROM firmware (BIOS).

Figure 1 shows the scope of Platform Secure Boot as it relates to the UEFI secure boot.

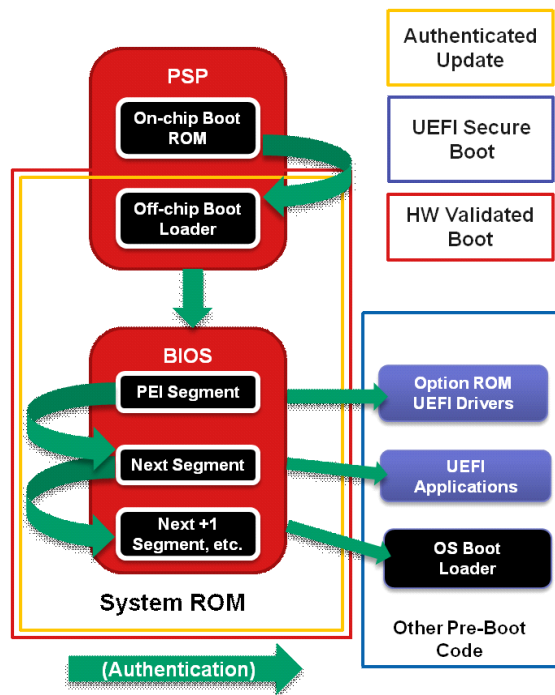


Figure 1. Platform Secure Boot Overview

The idea behind AMD Platform Secure Boot is to build a trusted boot environment even before starting the main cores. In the Platform Secure Boot mode, the PSP subsystem is the core root of trust for measurement.

During cold boot and under Platform Secure Boot, the PSP runs its own firmware. All of the main cores are held in the reset state while PSP firmware performs basic initialization, do the DRAM training and authenticates the main core reset code (i.e., the first block of BIOS). PSP firmware searches for this fraction of the BIOS image in the PSP directory and validates its signature. After validating the BIOS signature, the PSP firmware copies BIOS reset codes from SPI to DRAM, and configures the necessary hardware registers to release the main cores.. The main cores, upon reset,

start execution of the BIOS code authenticated by PSP firmware from DRAM. The BIOS maintains this trust chain by first authenticating all firmware components before passing control to these firmware components. UEFI Specification 2.3.1, Chapter 27, provides further guideline for BIOS expected behavior. This document does not discuss those details and the BIOS writers are expected to follow those guidelines in addition to the ones listed later in this document.

On resume from sleep, PSP firmware restores the memory controller, validates resume vector and releases the main cores. Once released, the x86 cores fetch code straight from Dynamic Random Access Memory (DRAM) based on boot time BIOS configuration. Chapter 5, “X86 BIOS S3-Resume Path Handling” on page 87, covers details regarding the BIOS-PSP information exchange to make this execution flow possible with X86 architecture.

For implementation details, refer to *Enabling Platform Secure Boot for AMD Family 17h Processor Based Client Platforms User’s Guide*, order# 56654 and *Enabling Platform Secure Boot for AMD Family 17h Models 00h–0Fh and 30h–3Fh and Family 19h Models 00h–0Fh Processor-Based Server Platforms*, order# 56534.

2.2 Firmware TPM Functions (Client PSP Only)

The PSP software solution-stack offers firmware-based TPM 2.0 services based on the Microsoft whitepaper, “Trusted execution environment ACPI profile.” BIOS writers are expected to follow the guidelines and provide BIOS support as outlined in that whitepaper. BIOS must wait for memory to be available before sending firmware-trusted platform module (fTPM) commands to PSP.

The PSP subsystem could use the dedicated SPI-ROM or share storage space with BIOS, based on the OEM platform design. If it chooses to use share storage space with BIOS, PSP relies on BIOS to provide the storage services to PSP firmware. The PSP firmware uses BIOS system management mode (SMM) mailbox services to save PSP data in SPI space. The PSP firmware encrypts the data block and uses BIOS runtime SMM handler services to store or update this data to SPI flash storage. The BIOS is expected to (a) reserve part of SPI flash region for PSP data storage, (b) provide services to PSP firmware to store and update PSP data to this SPI region, and (c) protect this region of SPI flash from writing by unauthorized code (using the chipset-provided flash locking mechanisms and Secure Flash Update). This region must be protected from any updates including BIOS updates. PSP firmware is expected to manage any updated TPM data within its own local memory until BIOS makes those storage services available to PSP firmware; in other words, the BIOS storage services are not expected to be available during early boot and resume path and PSP firmware is expected to not rely on BIOS storage services during that time.

Separately, BIOS can use the firmware TPM services for BIOS measurements as outlined in TCG specifications. In this usage model, the BIOS replaces the discrete-TPM PEI/DXE TIS driver with a firmware-based-TPM PEI/DXE driver; and BIOS exposes the TPM protocol defined in "TCG EFI protocol" specification. Also BIOS is expected to use the TPM2.0 command-set to communicate with integrated Trusted Platform Module (fTPM).

Chapter 3 PSP Components

PSP components are described in the following subsections.

3.1 On-chip PSP Boot ROM

Valid Program: All

PSP Boot ROM is an immutable part of the SOC and it embeds in it a SHA-256 hash of the public part of the AMD signing key, forms the hardware core root of trust for the Platform Secure Boot process. PSP microcontroller (A5) starts executing the On-chip Boot ROM code, in secure mode. It loads the off-chip PSP firmware into PSP static random access memory (SRAM) and after authenticating the PSP off-chip firmware it passes control to it. For Client AMD Family 17h Models 60h-6Fh and Server AMD Family 19h Models 10h-1Fh onward, PSP Boot ROM also supports the firmware anti-rollback feature, where control is passed to PSP off-chip firmware when the anti-rollback condition is passed.

3.2 Off-chip PSP Boot Loader

Valid Program: All

When PSP on-chip Boot ROM transfers control to PSP off-chip Boot Loader, it communicates the pre-loaded PSP Directory table address in PSP SRAM, in mailbox area at a pre-defined address within PSP SRAM.

3.2.1 AMD Family 17h Models 00h–0Fh Processor PSP Boot Loader

Hereafter, ERROR state comprises the actions of PSP writing the error condition code (post code) to FCH I/O port 80h, writing the error condition code to the FW_STATUS register, and entering a halt loop and no further forward progress is expected.

The off-chip PSP Bootloader is itself loaded and validated (and deflated and decrypted if required) by the on-chip BootROM. The process delineated below commences after the BootROM has performed this step and copied the PSP bootloader from the SPIROM to address 0 in SRAM, and sets the ARM[®] PC at that reset location.

All binary image validation and verification of signatures follow a common procedure unless otherwise noted:

PSP loads the image (usually from SPI-ROM) into SRAM and verifies its authenticity by calculating the hash of the FW, reading the signature from the loaded bin, and validating the signature of RSA-PSS using the AMD Root Signing RSA Public Key (unless another key is used, which will be noted). Hereafter, VALIDATE denotes the above process with any deviations noted.

1. Upon bootloader startup the following high level activities are performed:

- a. I-Cache and D-Cache and MMU are disabled if enabled by Bootrom. Various other HW Cortex-A5 related registers are programmed and setup appropriately.
- b. Configures the Translation Table related registers. Performs the setup of all page tables across the entire address space for all SRAM and DRAM. Note that both L1 and L2 page tables are used.
- c. Enables MMU; all addresses are now virtual as configured in item b.
- d. Branches to virtual entry point:
- e. Enters each mode used to disable interrupts and sets up the corresponding stack pointer
- f. From virtual entry point, sets the SVC mode stack, and branch to the main PSP bootloader entry point.
2. Within main bootloader entry point: Performs mapping of various PSP reserved regions, acquires some platform hardware specific information from the BootROM, determines the system core/die/socket configuration, configures required interrupts and handlers, and completes some initial hardware and die communication initialization.
3. [Slave die only] Prepares communication method with master die, and initialize mailbox registers.
4. [Master die only] Initializes the SPIROM for read/write/erase capability for either PSP SPI, BIOS SPI, or (rarely) both, and shares information with any slave die present.
5. Determines the current boot mode entered. Currently, S5 Cold/S5 Warm/S4/S0 are considered '(S5) Cold Boot' and S3 Resume is considered 'S3 Warm Boot'. Any other reported ACPI state from hardware is taken as an error and ERROR state is entered.
6. [Multi-die only] WAFL inter-die and inter-socket communication is configured, encryption is setup, and enabled.
7. The master die will establish the shared secret with any slaves for DRAM inline-AES encryption using Diffie-Hellmann, and broadcasts to any slaves; it will then wait for confirmation from the slaves.
8. Locates and loads the Security Gasket binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. Control is passed back to PSP after its execution, and a mode switch to Supervisor is made.
9. Checks for early secure debug unlock token, and handles request to enter into Debug Unlock mode.
10. Performs Stage 1 of the One-Time Programming (OTP) Fuse programming of certain fuses, including the anti-rollback counter, which is PSP auto-sense, and not based on external (BIOS) requests.

11. Locates and loads the IP Configuration binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise PSPs posts that the information contained in the bin is loaded, and continues.
12. Writes IP block information for MBAT table contents with MCA addresses for RAS usage.
13. Locates and loads the SMU FW (SMUFW0 for MP1) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in MP1 (not MP0) SRAM, and MP1 is taken out of reset. No loss of control on MP0 side unlike other bins loaded and executed in USR mode on MP0 SRAM by MP0 A5.
14. Waits for SMUFW to report that SMU (MP1) is ready. PSP continues.
15. Performs the milestone counter reset sequence as the preliminary step for eventual core complex and core release of x86 cores in the future for first fetch of BIOS instructions from DRAM.
16. Locates and loads the AGESA Bootloader Phase 1 (ABL1) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. The ABL1 will make various Supervisor mode (SVC) calls via ARM software interrupts (SWI) into PSP to perform any A5, or various other SOC-15 related activities.
17. The ABL1 will ask PSP to load (but not validate) APCB data into a location in SRAM of its choosing (determined from the APCB header information) from SPIROM.
18. The ABL1 is responsible for ABL initialization and setup for the forthcoming other ABL blocks which have compartmentalized features.
19. The ABL1 will ask PSP to load ABL2: PSP Locates and loads the AGESA Bootloader Phase 2 (ABL2) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. The ABL2 will make various Supervisor mode (SVC) calls via ARM software interrupts (SWI) into PSP to perform any A5, or various other SOC-15 related activities.
20. The ABL2 will ask PSP to load and validate S3 Resume APOB data into a location in SRAM of its choosing (determined from the APOB header information) from SPIROM. This will be used by ABL3.

21. The ABL2 will ask PSP to load and validate APPB data into a location in SRAM of its choosing (determined from the APPB header information) from SPIROM.
22. The ABL2 will perform many Data Fabric and related block initialization and configuration for system wide (including PSP and BIOS's) use.
23. The ABL2 will ask PSP to load ABL3: PSP Locates and loads the AGESA Bootloader Phase 3 (ABL3) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. The ABL3 will make various Supervisor mode (SVC) calls via ARM software interrupts (SWI) into PSP to perform any A5, or various other SOC-15 related activities.
24. The ABL3 may ask PSP to load and validate APCB data again.
25. The ABL3 is responsible for initializing, configuring, and enabling memory controller operation. Finally DDR memory is trained and ready for use via Data Fabric.
26. The ABL3 will ask PSP to load ABL4: PSP Locates and loads the AGESA Bootloader Phase 4 (ABL4) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. The ABL4 will make various Supervisor mode (SVC) calls via ARM software interrupts (SWI) into PSP to perform any A5, or various other SOC-15 related activities.
27. The ABL4 will ask PSP to load ABL5: PSP Locates and loads the AGESA Bootloader Phase 5 (ABL5) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bin is copied into the appropriate location in SRAM, a mode switch to user mode (A5 USR) is made, and the program counter is set to the start address for that bin in SRAM.
 - c. The ABL5 will make various Supervisor mode (SVC) calls via ARM software interrupts (SWI) into PSP to perform any A5, or various other SOC-15 related activities.
28. The ABL5 will setup data in the CCX (formerly CC6) CPU/System state configuration area via an SVC call to PSP.
29. The ABL5 will ask PSP to initialize various security and encryption features on the die.
30. Upon completion of ABL5, control is passed back to PSP after its execution, and a mode switch to Supervisor is made.
31. Executes 'Post DRAM Training Tests' to ensure access to DRAM is operating normally.

32. [Master die only. Note similarity to Step 12, but to DRAM without execution, and both SMUFW0 and SMUFW1]. Locates and loads the SMU FW (SMUFW0 and SMUFW1 for MP1) binary header from SPIROM, and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise the bins are copied into the appropriate location in DRAM, and PSP continues execution.
33. Informs SMU (MP1) via SMU-PSP mailbox registers of the following PSP information items for its use:
 - a. WAFL configuration information.
 - b. Shared DRAM address from Step 32.
 - c. Suitability to begin scan capabilities.
34. [Master die only] Prepares the shared inline-AES key for secure storage.
35. Locates and loads MP5 FW binary header from SPIROM and performs VALIDATE operation.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise, the bins are copied into the MP5 SRAM, and PSP continues execution.
36. [Master die only] Begins preparation for performing the validation and loading of BIOS from SPIROM to DRAM.
 - a. Obtains the BIOS OEM Public Key from SPIROM and copies to SRAM.
37. [Master die only] Locates and loads the BIOS binary header from SPIROM, and verifies its authenticity by calculating the hash of the FW, reading the signature from the loaded bin, and validating the signature of RSA-PSS using the OEM Public Key.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise, execution continues.
38. [Master die only] Locates and loads the BIOS Reset Image (if present) binary header from SPIROM, and verifies its authenticity by calculating the hash of the FW, reading the signature from the loaded bin, and validating the signature of RSA-PSS using the OEM Public Key.
 - a. If the signature verification fails, PSP enters ERROR state.
 - b. Otherwise, execution continues.
39. [Master die only] Copies the BIOS Reset Image from its location in SPIROM to its designated location in mapped DRAM space (the address, size, and other particulars are part of the BIOS image header).
 - a. Decompresses/Z-lib deflates the image in tandem to with loading, if required.
 - b. The copy (as all copies to DRAM and some SRAM locations) are done using an AMD safe copy routine for certain small sizes, or done using DMA via the CCP5's DMA engine.
40. Sets the S3 exit state (PMREG_INITPKG0) as preparation for core release (see also Step 14). Note that no CCX/CC6 data is accessed or written — this is done by ABL in Step 28.

41. The BIOS to PSP mailbox is initialized in preparation for communication (non-SMI, not in SMM) with BIOS upon entering either the steady state or starting the Trusted OS and trustlets.
42. [Master die only] Begins release of all populated/present/non-downcored Zen x86 cores on all populated core complexes. Notify all present slaves when cores have been released.
43. [Slave die only] Waits for notification from master die that all Zen cores have been successfully released.
44. Optionally, load the Diag Bootloader (part of the bootloader SPIROM image loaded into SRAM by on-chip BootROM).
45. PSP continues with execution separately for either the SP3, SP4, and SP5 processors or Socket AM4 processor. Henceforth, the SP4 processor will be treated the same as the SP3 processor. The distinction in bootloader operation and responsibilities is thus only between the Socket AM4 processor and the SP3 processor.
46. [Socket AM4 processors only] Loads the HMAC and inline-AES keys to PSP Bootloader mailbox region (4KB page at the end of SRAM) so that the Secure OS and mini-Bootloader does not need to re-derive this and other information.
47. [Socket AM4 processors only] Determines the current boot mode (as in Step 6), and proceeds according to current ACPI state.
48. [Socket AM4 processors only] For S5 Warm/S5 Cold (this process overwrites the PSP bootloader in SRAM, and never returns):
 - a. Validates and Loads the Secure OS:
 - b. Loads the FW header for the Secure OS image to determine its size, and in future the split address between Nwd and Swd.
 - c. Disables MMU, I-Cache and D-Cache, and return the stack pointer to its physical address.
 - d. Decompresses (if necessary) and safely copies Secure OS image to SRAM such that second bootloader ends right before the SVC stack starts at required physical address.
 - e. Disables interrupts, remap exception vectors, and jump to second-bootloader, which will copy Swd to address 0 and Nwd to 3_4000h. Then SBL will jump to Swd.
 - f. PSP Bootloader has been overwritten, control is passed to Swd and Secure OS.
49. [SP5 processors only] Initializes MPDMA engines, where PSP FW locates, loads, and VALIDATES MPDMA FW into respective MPDMA SRAM memories. (MPDMA is available on all AMD Family 19h Models 10h-1Fh devices.)
50. [Socket AM4 processors only] For S3 Resume:
 - a. Performs checking of stack pointer, disables interrupts, remaps the IVT, and other items.
 - b. Performs a jump to the mini-bootloader, and never returns.
51. [SP3 and SP4 processors only] Initializes the Secure Encrypted Virtualization (SEV) and Secure Nested Paging (SNP) subsystems in preparation for SEV/SNP calls from the host driver and hypervisor. (SEV is available on all AMD Family 17h series and Family 19h Models 20h-2Fh,

and Family19h Models 00h–0Fh devices. SNP is available on AMD Family 19h Models 20h-2Fh , Family19h Models 00h–0Fh, and later.)

52. [SP3 and SP4 processors only] Enters the PSP bootloader steady state, Wait-for-Interrupt (WFI) ultra-low power state (infinite loop), and remain there indefinitely until a power state transition:
 - a. The WFI steady state loop will remain in the ARM defined low power state after executing the WFI instruction until an interrupt occurs. The interrupt will be from one of the sources that are handled:
 - i. Checks for and handles the Early Secure Debug Unlock command from the JTAG to PSP (J2P) message mailbox registers, and places the system in Debug Unlock mode if required. For AMD Family 17h Models 3Fh–3Fh, Family 19h Models 00h–0Fh, and Family 19h Models 00h-0Fh there is a new type of debug unlock for mega data centers called Mega Center Secure Debug Unlock (MCSDU). This is also checked here in the program flow.
 - ii. Checks for any BIOS to PSP commands, and dispatches them for handling if required.
 - iii. Checks for any SMU to PSP commands, and dispatches them for handling if required.
 - iv. Checks for any Stage 2 OTP Fuse package programming requests from BIOS.
 - b. After the interrupt is received, interrupts and disabled, the interrupt is handled as above, and interrupts are re-enabled and the WFI loop is re-entered again to await another interrupt.
 - c. Processes 100 ms timer ticks without an external interrupt from the above sources.
 - d. PSP remains in the loop.

3.3 Off-chip PSP Secure OS for Family 17h

Valid Program: AMD Family 17h Models 00h–0Fh Processors and later

Please refer to Section 3.2.1 on page 25, Steps 44–50 for additional information.

The off-chip PSP Boot Loader will be overwritten in SRAM by the Secure/Trusted OS during initial boot up, near the end of the sequence as outlined in Section 3.2.5. Transfers of control to PSP Secure Operating System is then implicitly accomplished by setting the PC to the appropriate location in SRAM of the Swd image; it communicates certain state information such as the ACPI Sleep states mentioned in Section 3.2.5 Step 6, fTPM state (optional) and the PSP Bootloader mailbox referred to in Section 3.2.5 Step 45 at a pre-defined address with in PSP SRAM.

The same boot mode detection performed in PSP Bootloader is performed when the PSP Secure OS starts execution; it first determines if the system is booting from S5 or resuming from S3 (optional) state by reading the Sx state variable in the SRAM mailbox address.

PSP Secure OS performs the following sequence of operations as part of the S5 boot:

1. Performs the necessary initialization of OS internal structures and instantiates the TPM 2.0 compliant fTPM as a trusted application
2. Sets-up CPU-PSP interface registers' access control policy and interrupt mechanism

3. Re-initializes BIOS-PSP mailbox interface
4. Waits for BIOS SMM environment to be set up and BIOS to notify the SMM space reserved for PSP and parameters needed for PSP to generate SMI
5. The HMAC and inline-AES keys are obtained and unwrapped from their location in the PSP bootloader mailbox region.
6. Enters steady-state idling and waiting for commands from host interfaces

In steady state, when the system begins to enter S3 state, as shown in Figure 6, on page 72 PSP Secure OS is notified. Upon receiving this notification, PSP Secure OS prepares to enter the S3 state.

3.4 PSP AGESA™ Binaries

The "AGESA PSP binary" known as AGESA Boot Loader (ABL) is a single or set of binary images that is executed by the PSP. It is responsible for initializing APU silicon components (including but not limited to APU memory interface) on S5, S4 and S3 prior to the release of the main cores.

There may be one or more "AGESA PSP binary" that is responsible for a specific stage of the AGESA PSP initialization process. Each image has a separate entry in the PSP that must be populated by the platform BIOS.

Note: Refer to AMD Generic Encapsulated Software Architecture (AGESA™) V9 Interface Specification, order# 55483 for details

Chapter 4 Overview of BIOS Support for PSP

4.1 BIOS Build Process

The following subsections illustrate how the BIOS build process is updated to support a system with PSP.

4.1.1 Build SPI Image

In the case of SPI-ROM, an image consisting of all the PSP firmware components and System/BIOS firmware components is created as part of the BIOS build process and is programmed into the SPI-ROM device. The image also contains PSP directory tables that describe the various firmware components.

The BIOS needs to reserve a region in boot media to hold the location of the PSP and BIOS Directory Table as well as the contents of PSP and BIOS Directory table. Also Embedded Firmware Structure that on-chip PSP Boot ROM will scan for the PSP Directory address needed to put the correct address.¹

During the BIOS build, the "BuildPspDirectory" Tool will be called to build the Directory Header, and automatically insert the content described by a configuration file. Refer to Appendix B on page 125 for details of the configuration file's format.

For the two directory table case, two configuration files need to be prepared, and the "BuildPspDirectory" tool will be called twice to build two separate Directory Blobs.

Notes:

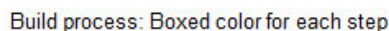
1. Refer to Table 2. "Embedded Firmware Structure" on page 37 for address requirements
2. Refer to Appendix B on page 125 for details.

Additionally, note that all Address fields used in the SPI case are physical addresses.

4.1.2 BIOS Build Flow

As shown in Figure 2 on page 34, the SPI image includes BIOS components as well as PSP components. The BIOS PEI volume is considered the BIOS RTM volume.

To support Platform Secure Boot, the RTM volume resides in the BIOS directory needs to be signed and the PSP directory and BIOS Directory needs to be present to provide information regarding various signed entities. Figure 2 on page 34 is a diagram that summarizes the above discussion to illustrate how various entities listed above can be combined to build the final SPI image, especially how BIOS RTM signature has been made.



- As shown above, the first step is generation of the digital signature of public portion of the OEM signing key. This may happen just once before the start of project.
 - The OEM/IBV submits the public key portion of their OEM signing key to AMD.
 - AMD signs this key using the AMD RSA key and passes back it to IBV/OEM.
 - The AMD public key and the signed OEM public key are part of final BIOS SPI image.
 - Next, the BIOS source code is compiled and various BIOS components (PEI Volume, DXE volume, NVRAM storage, EC binary, etc.) are built as usual.
- As part of the build process, to extend the trust chain, it is suggested to have PEI authenticate the DXE volume.
- The PSP directory and BIOS directory are built next. PSP directory and BIOS directory table points to the location of various firmware entities.
- BIOS binaries, PSP directory, BIOS directory and various firmware binaries are combined to build the SPI BIOS image.

- Finally, the OEM signing server builds the signed BIOS RTM signature based on blob of BIOS PEI volume concatenated with BIOS Directory header, and generates the digital signature of this using private portion of OEM signing key. The SPI location for signed BIOS RTM code is finally updated with this signature blob.

Note: Any signature generation tool is acceptable for this step. Commonly used tools include *openssl* and *signtool*. Byte order must be reserved if the signature is generated in big-endian format using a tool like *openssl*.

After the above steps, the final SPI BIOS image will be ready.

4.1.2.1 Signing of BIOS Component—OEM Signing Key, RTM Volume

When the OEM enables the Platform Secure Boot, OEM must sign the BIOS RTM volume using the private portion of secure RSA key. This key used to sign the BIOS RTM volume is referred to as the OEM signing key. OEM keeps the private portion of the OEM signing key in a secure place (HSM, etc.) and submits the public portion of OEM key to AMD. AMD performs one time signing of public portion of OEM signing key. This process enables PSP firmware to authenticate OEM public key.

On secure PSP parts, the PSP firmware authenticates the BIOS image in two steps before releasing x86 core. PSP firmware first parses PSP BIOS directory to locate the signed OEM public key and authenticates the OEM public key that was signed with AMD signing key. Next after the public portion of OEM signing key is authenticated, the PSP firmware uses the OEM public key to further authenticate the BIOS RTM volume that was signed by OEM secure private key.

If the signature matches, the BIOS is considered trusted and x86 cores are released.

One note about the signed BIOS RTM volume: The signed BIOS blob is generated by first concatenating BIOS RTM volume with BIOS directory blob and signing this combined blob using private portion of OEM key. Integrity of both BIOS directory and RTM volume integrity can now be checked together when PSP firmware authenticates this blob. This combined blob must be signed with the BIOS Signing RSA Private Key using the RSASSA-PSS signing scheme used as signature scheme with SHA-256 used as the hashing algorithm for both message and mask generation. The resulting signature data is stored in the PSP directory entry as the entry type 0x07. The size of the signature data will be 256 bytes for 2048-bit key or 512 bytes for 4096-bit key

This two-step authentication removes unnecessary dependence on AMD signing server by build processes where BIOS is built on regular basis by IBV/OEM. AMD signing server will sign the public portion of OEM signing key once at the beginning of project and separated from the BIOS build process; during the normal BIOS build process the private portion of OEM signing key will be used to sign BIOS RTM volume as part of OEM build process without any AMD signing server involvement. The BIOS image includes the AMD public key as well as signed public OEM signing key in the PSP image that was generated at the beginning of the project. This allows the OEM to use internal signing processes without external dependency.

After the PSP firmware releases host processor for execution, the BIOS is expected to maintain the chain of trust to authenticate next set of BIOS code before executing it. It is left to OEM/IBV to choose appropriate BIOS implementation to insure the trust chain. At x86 core release the RTM

volume has been authenticated by PSP firmware. BIOS RTM volume must authenticate the next volume before handling of the control. If only the SEC code is the BIOS RTM volume, then SEC code must authenticate PEI volume before handing off control to PEI core. If BIOS RTM volume is entire PEI volume it must authenticate DXE volume. The DXE IPL code in BIOS PEI volume needs to further authenticate DXE volume before handing off control to DXE code.

AMD will provide the signed OEM public key and signed AMD public key. The format of AMD signing key and OEM signing key is shown in Appendix B on page 113.

4.1.3 Directory Table

A PSP directory table is needed to aid PSP firmware in finding various components in the offchip boot storage media. This PSP directory is a simple table of consist of various entries. Each entry provides information about various firmwares in offchip boot media such as their type, size and location. The PSP directory can be anywhere in the SPI storage. Embedded Firmware Signature (EFS) which used to point to the PSP Directory should put at a fixed address.

The Embedded Firmware Structure is used to provide the location of the PSP Directory and some SPI information updated by AGESA-FCH during post time. Table 2 describes the embedded firmware structure field.

The PSP on-chip firmware scans the following offset for the first instance of the Embedded Firmware Structure in the 16M SPI address window.

- 1st Address checked (recommended) 0xFA0000
- 2nd Address checked 0xF20000
- 3rd Address checked 0xE20000
- 4th Address checked 0xC20000
- 5th Address checked 0x820000
- 6th Address checked 0x20000

For server programs, Family 19h Models 10h-1Fh, Embedded Firmware Structure search order supports only one EFS offset, at 0x20000, in the respective 16 Mbyte page (four pages in the 64 MByte SPIROM).

For client programs, Family 19h Models 40h-4Fh, Embedded Firmware Structure search order changes to:

- 1st Address checked (recommended) 0x20000
- 2nd Address checked 0x820000
- 3rd Address checked 0xC20000
- 4th Address checked 0xE20000
- 5th Address checked 0xF20000
- 6th Address checked 0xFA0000

The PSP identifies the Embedded Firmware Structure by the signature of 0x55AA55AA at the start of the structure. The Embedded Firmware Structure contains data that is required to boot the system, such as the address of the PSP directory. Care must be taken by the system designer in choosing the systems ROM layout to ensure that the PSP always finds the intended Embedded Firmware Structure first.

Recommendation:

For server Family 19h Models 10h-1Fh: Only one Embedded Firmware Structure is supported, at offset 0x20000.

For all server family legacy devices prior to server Family 19h Models 10h-1Fh: Always place the Embedded Firmware Structure at address 0xFA0000 in the system ROM.

For client programs: Always place the Embedded Firmware Structure at the address at the BootROM first searched in the system ROM. This ensures the PSP always locates the intended Embedded Firmware Structure first.

Alternative recommendation: Place the Embedded Firmware Structure at one of the allowed addresses, and ensure the pattern 0x55AA55AA does not occur at any of the addresses checked by the PSP prior to reaching the intended Embedded Firmware Structure. System designers are warned against placing UEFI variables, logs, configuration settings, user supplied graphics images, or similar system or user supplied data at one of the addresses the PSP would check while looking for the intended Embedded Firmware Structure. Starting from AMD Family 19h series, this recommendation is not critical because each EFS occurrence is matched against a Multi Gen EFS value. Refer to “EFS Search Algorithm” on page 60.

Table 2. Embedded Firmware Structure

Offset (Hex)	Size (Bytes)	Description/Purpose
0x00	4	Signature of Embedded Firmware Structure (0x55AA55AA)
0x14	4	Pointer directly to PSP Directory table start from Family 17h Models 00h-0Fh, or point to PSP combo Directory header. For combo directory, reference Section 4.1.4.2 for details
0x18	4	Pointer to BIOS Directory table for Family 17h Models 00h-0Fh. Refer to Table 15 on page 55 for details of BIOS directory table.
0x1C	4	Pointer to BIOS Directory table for Family 17h Models 10h-1Fh. Refer to Table 15 on page 55 for details of BIOS directory table
0x20	4	Pointer to BIOS Directory table for Family 17h Models 30h-3Fh. Refer to Table 15 on page 55 for details of BIOS directory table and Family 17H Model 70h-7Fh.

Table 2. Embedded Firmware Structure (Continued)

Offset (Hex)	Size (Bytes)	Description/Purpose
0x24	4	<p>For AMD client products, starting with Family 17h models 30h-3Fh:</p> <p>Bit 0 Second_gen_efs:</p> <ul style="list-style-type: none"> • If 1, this is a first-generation EFS structure that is ignored by the PSP Boot ROM. • If 0, then this is a second generation EFS structure. <p>For AMD server products, starting with Family 19h models 00h-0Fh:</p> <p>0x24[15:0]: A bit that is clear in this 16-bit field indicates that the EFS is compatible with generations of processors checking that bit. Multiple bits may be clear, which indicates compatibility of an EFS with multiple processor generations.</p>
0x28	4	<p>Beginning with AMD Family 17h Model 60h and AMD Family 19h Model 0h, this is the Address pointer to the BIOS directory or the Combo directory structure.</p> <p>The PSP FW will perform these steps to load the BIOS directory from the SPI.</p> <ol style="list-style-type: none"> 1. Check whether the EFS offset 0x28 points to the BIOS directory (validate against the unique cookie/sign and checksum). If it does, copy the BIOS directory to the SRAM space and exit. Otherwise, perform step 2. 2. Check whether the EFS offset 0x28 points to the Combo directory (validate against the unique cookie/signature and checksum). If it is a valid Combo structure, perform step 3. Otherwise, exit with an error. 3. Read the ID from each Combo BIOS entry and compare it with the PSP ID. <ul style="list-style-type: none"> • If the ID matches, copy the address of the BIOS directory. • Validate the BIOS unique signature and checksum. • If they are valid, copy them to SRAM. <p>For more information about PSP ID, please refer to Table 12., Valid PSP IDs per Program.</p> <p>For more information about the Combo BIOS directory, refer to 4.1.4.2, Combo Directory BIOS Support.</p>
0x2C	4	Family 19h Models 40h-4Fh onward, backup copy of the PSP L1 directory if A/B recovery enabled.
0x30	4	Pointer to promontory firmware.
0x34	4	Pointer to low power promontory firmware.

Table 2. Embedded Firmware Structure (Continued)

Offset (Hex)	Size (Bytes)	Description/Purpose
0x40	1	<p>SpiReadMode for AMD Family 15h Models 60h-6Fh (update by AGESA-FCH)</p> <p>000b Normal read (up to 33M)</p> <p>001b Reserved</p> <p>010b Dual IO (1-1-2)</p> <p>011b Quad IO (1-1-4)</p> <p>100b Dual IO (1-2-2)</p> <p>101b Quad IO (1-4-4)</p> <p>110b Normal read (up to 66M)</p> <p>111b Fast Read</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Micron SPI chips are not supported. 2. To avoid AGESA™ to update the field SpiReadMode/FastSpeedNew <ol style="list-style-type: none"> 1. Predefine SpiReadMode = PcdResetMode, either 101b Quad IO (1-4-4) or 111b Fast Read according board SPI design . 2. Predefine FastSpeedNew = (PcdResetFastSpeed-1)
0x41	1	<p>FastSpeedNew for Family 15h Models 60h-6Fh</p> <p>Value Description:</p> <p>0000b 66.66MHz.</p> <p>0001b 33.33MHz.</p> <p>0010b 22.22MHz.</p> <p>0011b 16.66MHz.</p> <p>0100b 100MHz.</p> <p>0101b 800kHz.</p>
0x42	1	Reserved

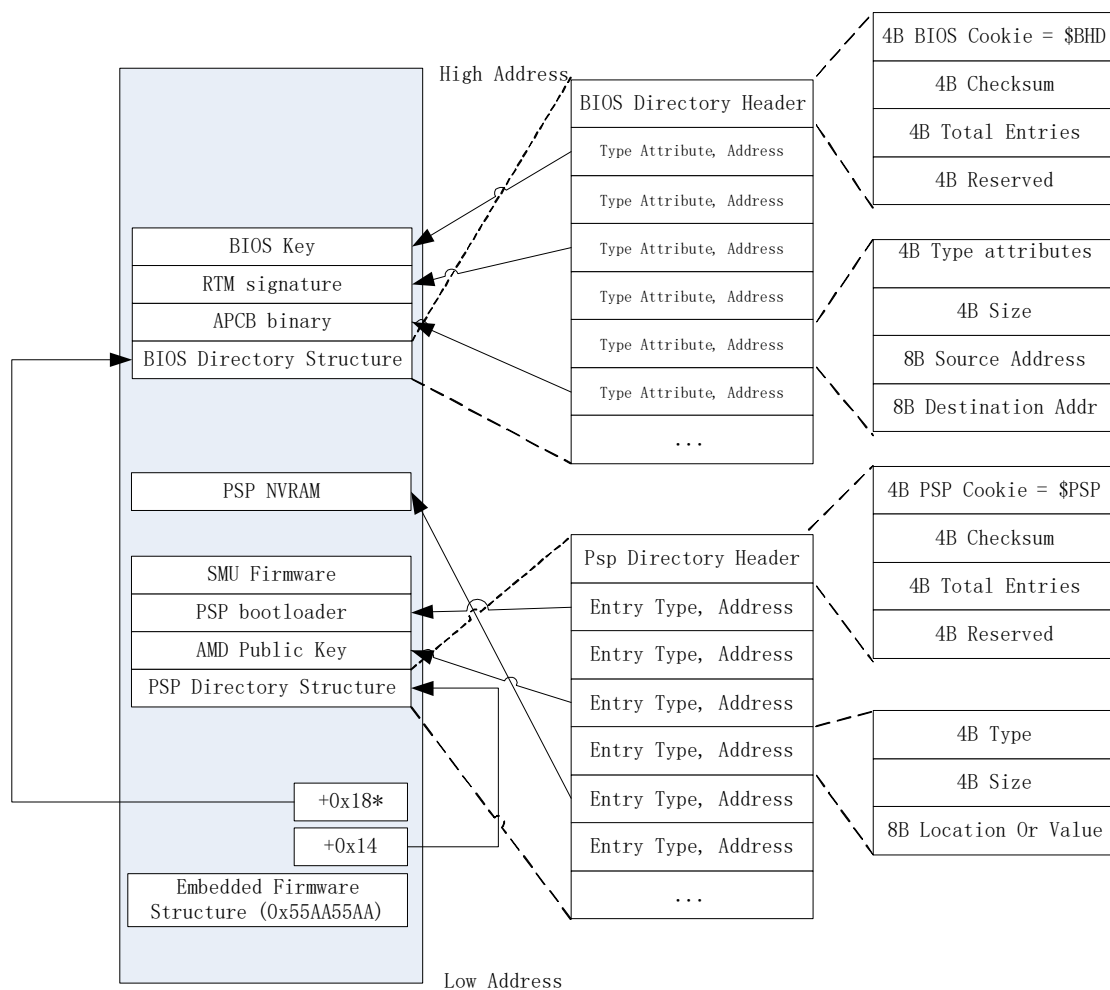
Table 2. Embedded Firmware Structure (Continued)

Offset (Hex)	Size (Bytes)	Description/Purpose
0x43	1	<p>SpiReadMode for Family 17h Models 00h-0Fh,10h-1Fh (update by AGESA-FCH)</p> <p>Value Description:</p> <p>000b Normal read (up to 33M)</p> <p>001b Reserved</p> <p>010b Dual IO (1-1-2)</p> <p>011b Quad IO (1-1-4)</p> <p>100b Dual IO (1-2-2)</p> <p>101b Quad IO (1-4-4)</p> <p>110b Normal read (up to 66M)</p> <p>111b Fast Read</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. If the board design may use Micron SPI chips and other makers's chips, 0xFF should be predefined for SpiReadMode/FastSpeedNew/QPR_DummyCyc to SPI compatibility. 2. To avoid AGESA to update the field SpiReadMode/FastSpeedNew/QPR_DummyCyc, <ol style="list-style-type: none"> 1.Predefine SpiReadMode = PcdResetMode, either 101b Quad IO (1-4-4) or 111b Fast Read according board SPI design . 2.Predefine FastSpeedNew = (PcdResetFastSpeed-1) 3.Predefine QPR_DummyCyc = 0x0A for Micron chip, 0xFF for other chip.
0x44	1	<p>FastSpeedNew for Family 17h Models 00h-0Fh,10h-1Fh</p> <p>Value Description:</p> <p>0000b 66.66MHz.</p> <p>0001b 33.33MHz.</p> <p>0010b 22.22MHz.</p> <p>0011b 16.66MHz.</p> <p>0100b 100MHz.</p> <p>0101b 800kHz</p>
0x45	1	<p>QPR_Dummy Cycle configure for Family 17h Models 00h-0Fh, 10h-1Fh,.</p> <p>Note: For SPI chips from Micron, it should be 0x0A. Otherwise it should be 0xFF</p>
0x46	1	Reserved

Table 2. Embedded Firmware Structure (Continued)

Offset (Hex)	Size (Bytes)	Description/Purpose
0x47	1	(ReadOnly) SpiReadMode for Family 17h Models 30h-3Fh and later Families if (SpiReadMode==2,3,4,5,6,7) {set spimode} else {do nothing} Value Description: 000b Normal read (up to 33M) 001b Reserved 010b Dual IO (1-1-2) 011b Quad IO (1-1-4) 100b Dual IO (1-2-2) 101b Quad IO (1-4-4) 110b Normal read (up to 66M) 111b Fast Read <i>Note: 1. It is EFS creator's responsibility to choose right SPI mode & FastSpeed for the mother board.</i>
0x48	1	(ReadOnly) SpiFastSpeed for Family 17h Models 30h-3Fh and later Families if (SpiFastSpeed==0,1,2,3,4,5) {set SpiFastSpeed} else {do nothing} Value Description: 0000b 66.66 MHz. 0001b 33.33 MHz. 0010b 22.22 MHz. 0011b 16.66 MHz. 0100b 100 MHz. 0101b 800 kHz.
0x49	1	(ReadOnly) MicronDetectFlag for Family 17h Models 30h-3Fh and later Families if (MicronDetectFlag==0x55) { if (PSP detect Micron VendorID ==0x20) { DPR_DUMMYCYCLE = 0x8; QPR_DUMMYCYCLE = 0xA;}} else if (MicronDetectFlag==0xAA) { DPR_DUMMYCYCLE = 0x8; QPR_DUMMYCYCLE = 0xA;} else {do nothing} Value Description: 0x55:force Micron detection, if the bios supports Micron and non Micron. 0xAA:force Micron dummymcycle, if the bios supports Micron only others: do nothing, if the board does NOT support Micron
0x4A	1	Reserved

Figure 3 on page 42 describes the layout of the PSP Directory Table in SPI-ROM and how it is used to locate various PSP related public key tokens, firmware images and corresponding signatures.



*0x18, The offset used to point to the BIOS directory is varies between multiple programs

Figure 3. PSP Directory Table

4.1.4 PSP Directory Table

Table 3 describes the fields of the PSP Directory Table header structure.

Table 3. PSP Directory Table Header Fields

Field Name	Offset (Hex)	Size (Bytes)	Description/Purpose
PSP Cookie	0x00	4	PSP cookie "\$PSP" to recognize the header.

Table 3. PSP Directory Table Header Fields(Continued)

Field Name	Offset (Hex)	Size (Bytes)	Description/Purpose
Checksum	0x04	4	32-bit CRC value of the header items below this field and the including all entries. Fletcher's checksum algorithm is used for CRC calculation.
Total Entries	0x08	4	Number of PSP Directory Table entries in the table.
Additional Info	0x0C	4	Additional directory information. See Table 4.

Table 4. PSP Directory Table Additional Info Fields

Field Name	Offset	Size (Bits)	Description
Max Size	0	10	Directory Size in 4K bytes
SpiBlockSize	10	4	Modifiable Entry alignment
Base Address	14	15	[26:12] of Directory Image Base Address
Address Mode	29	2	Directory Address Mode (0, 1, 2, or 3)

Table 5 describes the fields of the PSP Directory Table Entry structure, is followed by the PSP Directory Table Header.

Table 5. PSP Directory Table Entry Fields

Field Name	Byte Offset	Byte Length	Description
Entry	0	4	Describe the attributes of the PSP Entry, including Type, Subprogram, ROMId, Instance, and so on. Refer to Table 6 for detailed bit field definitions.
Size	4	4	Size of the PSP entry in bytes. Ignore if Entry.Type is equal to 0x0B.
Location	8	8	If Entry.Type is equal to 0x0B, treat as 64-bit value or treat as location. For location, refer to Table 7 for detailed bit field definitions.

Table 6. PSP Entry Bit Field Definition

Field Name	Bits	Description
Type	0:7	Type of PSP entry
Subprogram	8:15	Specify the SubProgram. For more information, see Section Table 4.1.6.

Table 6. PSP Entry Bit Field Definition (Continued)

Field Name	Bits	Description
ROMId	16:17	0: Content placed in the SPI device select by SPI_CS1_L 1: Content placed in the SPI device select by SPI_CS2_L. For details on how to declare an entry with ROMID, see appendix Section B.1.2, “Node <PSP_DIR>” on page 127.
Writable	18	0: Region read only. 1: Region writable, This field must be filled when the ROM Armor feature is enabled.
Instance	19:22	Instance of type
Reserved	23:31	Reserved

Table 7. Location Bit Field Definition

Field Name	Bits	Description
Address	0:61	62-bit address
Address Mode	62:63	00: X86 Physical address 01: offset from start of BIOS (flash offset) 02: offset from start of directory header 03: offset from start of partition For more information, see Appendix B, “BuildPspDirectory Tool Version 4.x” on page 125, Step f, Attribute: AddressMode (Optional) Same as image entry.

4.1.4.1 AMD Family 17h and 19h Processor PSP Directory Type Encodings**Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings**

PSP Entry Type	Description/Purpose
0x00	AMD public Key. Must be the first entry of the PSP directory.
0x01	PSP Boot Loader firmware or stage 1 bootloader if A/B recovery is enabled.
0x02	PSP Secure OS firmware
0x03	PSP Recovery Boot Loader
0x04	PSP Non Volatile data. In combo BIOS implementation, this region can be allocated to share between different program to save SPI footprint. For more information about managing the corruption of this entry, refer to 6.2.2, Clear fTPM NVRAM
0x08	SMU off chip firmware
0x09	AMD Secure Debug Key
0x0A	(OEM) ABL public key signed with AMD key

Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings (Continued)

PSP Entry Type	Description/Purpose
0x0B	PSP Soft Fuse Chain, 64-bit value field. For detailed definition, refer to Table 9 on page 49 for PSP Soft Fuse Chain definition. Declare as VALUE_ENTRY in BuildPspDirectory tool XML file.
0x0C	PSP boot loaded trustlet binaries
0x0D	Trustlet public key signed with AMD key
0x12	SMU off chip firmware
0x13	PSP early secure unlock debug image. Can be removed, if token unlock debug feature is not required on production image.
0x20	IP Discovery Binary. IP discovery information is consolidated into binary data that is delivered by PSP boot loader in to fixed DRAM location. The IP discovery binary data include IP list, base address and IP version number. It can be queried by software, which includes driver, tools, firmware and other software components.
0x21	Wrapped iKEK
0x22	PSP token unlock data. Can be removed, if token unlock debug feature is not required on production image.
0x24	Entry to load security policy binary
0x25	MP2 FW
0x26	Reserved for MP2 FW part2. (for Family 17h Model 10h-1Fh only)
0x27	User mode unit tests binary (for Family 17h Model 10h-1Fh only)
0x28	PSP Entry points to system driver in SPI space
0x29	Location field pointing to KVM image.
0x2A	Location field pointing to MP5 FW.
0x2B	Optional field, Physical address of Embedded Firmware Structure. Filled only when the Embedded Firmware Structure is not placed at the offset 0x20000 of SPI binary (only for Family 17h Models 10h-1Fh and Family 17h Models 60h-6Fh).
0x2C	TEE write once NVRAM.
0x2D	External Premium Chipset PSP Boot Loader
0x2E	External Premium Chipset MP0 portion of DXIO FW binary
0x2F	External Premium Chipset MP1 FW binary
0x30	PSP AGESA Binary 0
0x31	PSP AGESA Binary 1
0x32	PSP AGESA Binary 2
0x33	PSP AGESA Binary 3

Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings (Continued)

PSP Entry Type	Description/Purpose
0x34	PSP AGESA Binary 4
0x35	PSP AGESA Binary 5
0x36	PSP AGESA Binary 6
0x37	PSP AGESA Binary 7
0x38	SEV Data (for Server product only) Preserve this across BIOS update.
0x39	SEV Code (for Server product only)
0x3A	Processor serial number white list binary
0x3B	SERDES PHY microcontrollers microcode, initialized by DXIO firmware running in ABL.
0x3C	VBIOS pre-load for early initialization and GDB programming.
0x3D	WLAN Umac
0x3E	WLAN Imac
0x3F	WLAN Bluetooth
0x40	Point to 2nd level PSP Directory, for more information refer to section 4.5 PSP initiated Crisis Recovery Path
0x41	External Premium Chipset MP0 Bootloader
0x42	Separated DXIO PHY SRAM FW
0x43	Public key of Separated DXIO PHY SRAM FW Only applicable to Family 17h Models 30h-3Fh.
0x44	USB unified PHY FW Declare as POINT_ENTRY in BuildPspDirectory tool XML file.
0x45	Entry to load security policy binary for tOS
0x46	External Premium Chipset PSP Boot Loader
0x47	DRTM TA
0x48	(Client programs only) In A/B recovery schema, this entry is placed in L1 PSP directory, and points to the L2A PSP Directory header. Declare as POINT_ENTRY in BuildPspDirectory tool XML file.
0x49	(Client programs only) In A/B recovery schema, this entry is placed in L2A or L2B PSP directory, and points to L2 BIOS Directory header inside the same region (A or B). Declare as POINT_ENTRY in BuildPspDirectory tool XML file.
0x4A	(Client programs only) In A/B recovery schema, this entry is placed in L1 PSP directory, and points to L2B PSP Directory header. Declare as POINT_ENTRY in BuildPspDirectory tool XML file.

Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings (Continued)

PSP Entry Type	Description/Purpose
0x4B	Reserved
0x4C	Security Policy Binary for External Premium Chipset
0x4D	Secure Debug Unlock binary for External Premium Chipset
0x4E	PMU Public Key signed with AMD key Only applicable to Family 17h Models 30h-3Fh.
0x4F	Point to UMC FW
0x50	This entry points to the Public Keys Table (also referred as Key Data Base) for PSP Bootloader. Public keys are needed to certify firmware components and data. Public keys are stored in SPI ROM and are loaded into SRAM.
0x51	This entry points to the Public Keys Table (also referred as Key Data Base) for PSP tOS. Public keys are needed to certify firmware components and data. Public keys are stored in SPI ROM and are loaded into SRAM.
0x52	Entry points to OEM PSP BootLoader User Application [If supported by program] During early execution, PSP BL allows OEM applications to be executed from PSP BL. *
0x53	Entry points to OEM PSP BootLoader User Application Signing Key PSP BL allows OEM specific application to execute early during boot, and this key will be used to sign this OEM application.
0x54	Entry point to PSP NVRAM for RPMC. Refer to 6.2.2, Clear fTPM NVRAM for more detail on how to manage the corruption of this entry.
0x55	Table of SPL values used by bootloader for FW anti-rollback
0x56	Table of SPL values used by SecureOS for FW anti-rollback
0x57	CVIP configuration table, the information will be consumed by PSP BL
0x58	DMCU-ERAM (for Family 17h Model 60h-6Fh), is used for USB-C DP-alt mode switch. And it's possible to load at early stage, before Gfx is loaded, as their loading destination is internal IP SRAM.
0x59	DMCU-ISR (for Family 17h Model 60h-6Fh), is used for USB-C DP-alt mode switch. And it's possible to load at early stage, before Gfx is loaded, as their loading destination is internal IP SRAM.

Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings (Continued)

PSP Entry Type	Description/Purpose
0x5A	MSMU binary 0 Description: MSMU” (Middle SMU). Its purpose is to help improve the entry and exit latency during the CPUOFF and GFXOFF states. The MSMUs are implemented as micro sequences and require the use of FSDLs which are compiled into binaries. The MSMU requires two binaries to be signed and included in the BIOS ROM Consumer: PSP loads from SPI ROM into RSMU SRAM Authenticator: PSP Bootloader
0x5B	MSMU binary 1
0x5C	SPI-ROM configuration A binary contains information on multiple SPI-ROM vendors, such as necessary op-codes to enable quad-mode and op-codes for reading, writing, erasing, and so on.
0x5D	MPIO Offchip FW MPIO Offchip Firmware, responsible for xGMI, WAFL, PCIe, and other training.
0x5E	DF topology blob
0x5F	HSP TPMLite binary
0x71	PSP Entry for DMCUB (Family 17h Model 90h-9Fh) firmware instruction part This is used for PSP to load the DMCUB firmware instruction part into a trusted memory region during pre-OS.
0x72	PSP Entry for DMCUB (Family 17h Model 90h-9Fh) firmware data part This is used for PSP to load the DMCUB firmware data part into the frame buffer during pre-OS.
0x73	PSP Stage 2 Boot Loader firmware This is loaded by Stage 1 PSP BL. This entry is required for all products starting from Family 17h Model 90h.
0x74	PSP Platform Driver This is loaded by PSP Secure OS. The driver has features implemented specific to the platform. This is a new entry starting from Family 17h Models 90h-97h.
0x75	Firmware Soft Fusing Binary image Applicable to Family 19h Models 10h-1Fh only.
0x76	Register Initialization Binary (RIB)
0x80	OEM Sys-TA
0x81	OEM Sys-TA Signing Key
0x82	IKEK_OEM, wrapped using IKEK_AMD

Table 8. AMD Family 17h and 19h Processor PSP Directory Type Encodings (Continued)

PSP Entry Type	Description/Purpose
0x83	Table of SPL values used for FW anti-rollback for customer signed binaries. This entry is currently specific to Family 17h Model 98h-9Fh.
0x84	tKEK_OEM, wrapped using tKEK_AMD
0x85	AMF firmware part 1 Lx core can start execution only from SRAM. Part 1 FW will be used to initialize MPM TLBs so that DRAM can be mapped to Lx.
0x86	AMF Firmware part 2 Most of the MPM functionality will be implemented in this part.
0x87	MFD will configure the MPM- factory provisioning data in TEE Objects
0x88	WLAN firmware copied to MPM memory by MFD and then MPM will send this to WLAN
0x89	MPM functional driver catering to all security needs of MPM
0x8A	USP4 PHY Firmware. This firmware configures the USB4 PHY SRAM firmware.
0x8B	FIPS certification module. FIPS certification dedicated module used for FIPS certification of PSP.
0x8C	MPDMA TF FW
0x8D	iKEK_TA, iKEK_TA: iKEK key used to encrypt SOC agnostic firmware (e.g., Trusted Applications). iKEK_TA is wrapped with iKEK_SOC.
0x8E	Secure Firmware Data Recorder (sFDR). This module provides a methodology to record targeted SoC and/or FW data.
0x8F	dUSB4, This firmware is consumed by off-chip module to configure USB4.
0x91	GMI3 PHY firmware
0x92	Firmware for MPDMA Page Migration instances
0x93	PROM21 firmware. This firmware is consumed by an off-chip module to configure discrete XHCI/AHCI/PCIe.
0x94	LSDMA firmware. Manages the AMD HW IP block of LSDMA engines and handles the hardware initialization. Handles command execution, clock gating, power saving management, exception, and interrupt. Handles context management, like context switch and preemption. Monitor LSDMA status
0x95	C20 PHY FW, Separated DXIO C20 PHY SRAM FW

Table 9. PSP Soft Fuse Chain Definition

Bit Field	Description
0	Secure Debug Unlock. 0: Secured Debug Unlock disabled. 1: Secure Debug Unlock enabled.

Table 9. PSP Soft Fuse Chain Definition(Continued)

Bit Field	Description
1	Reserved
2	Early Secure Debug Unlock. 0: Disable early secure debug unlock. 1: Enable early secure debug unlock.
3	Token Unlock passed. BIOS sets this bit after it saves the Token in the NV RAM; in subsequent boot BL checks it and if it's there read the Token and unlock.
4	Enable applying of Security Policy to unsecure ASIC: 0 - normal boot w/o load security policy; 1 - load security policy regardless secure state.
5	Load Diag BL: 0 - normal boot; 1 - PSP BL loads Diag BL/ sBIOS disables fTPM, if non-secure part detected; otherwise ignore this bit.
6	Disable PSP Debug Prints: 0 - normal boot with debug prints following config; 1 - force disable PSP debug prints.
7-13	Reserved for PSP Debug.
14	SPI decode selection for 32 MB ROM. 0: decode the ROM 000000-FFFFFF to MMIO FF000000-FFFFFFFF, 1: decode the ROM 1000000-1FFFFFFF to MMIO FF000000-FFFFFFFF
15	Program with LPC support removed: Debug message output selection 0: SOC UART (e.g., 0xFEDC9000) 1: IO port 0x3F8 (On CRB, it is decoded by EC or UDC card, depending on system config.) Program with LPC support: Postcode decode selection 0: Postcode decode to LPC 1:Postcode decode to eSPI
16	Reserved for X86 HDTOUT Switch.
17	Reserved for ABL Console Out (Serial Out) Switch.
18	Reserved for ABL Breakpoint Switch.
19	Reserved for ABL HDTOUT Switch.
20-27	Reserved for ABL HDTOUT Die Bitmask.
28	MP0 DPM Enable, implemented in Family 17h Model 10h-1Fh.
29	Skip MP2 FW loading. 0: Load MP2 FW normally 1: Skip MP2 FW Loading.
30	Control output of Postcode in output 1-byte format. 1: enable
31	Legacy recovery support. 1: PSP bootloader from current partition simulates legacy recovery behavior, and only critical images will be loaded.
32	FIPS certification enablement: 0: FIPS certification mode is OFF; 1: FIPS certification mode is ON.
33	Force MPM enabled on MPM disable SKU in early program phase, disabled in the production Firmware.
34	Disable SPIROM-CONFIG feature This bit is consumed in PSP BL to control SPIROM-CONFIG feature disablement. 0: default – feature enabled, 1: feature disabled.
35-63	Reserved.

4.1.4.2 Combo Directory BIOS Support

To allow customers to reuse their system design, some of the AMD processors are packaged using the same socket. With this design, there is a requirement for BIOS to ask for one BIOS binary to support multiple processor families that are using same socket. Since every processor family uses a unique AMD root key, and PSP firmware, the SMU firmware and the BIOS OEM key need to sign with this unique root key, it requires having multiple PSP directory tables embedded to one BIOS binary.

Two-level PSP or BIOS Directory structures, 1st level with program, address pair, 2nd level with real PSP directory and a filed (offset 0x14 and offset 0x28) defined in “Embedded Firmware Structure” on page 37 are introduced to support this requirement. Starting from Family 17h Models 00h-0Fh, the PSP on-chip boot loader will look up the PSP directory pointer from Embedded Firmware structure offset 0x14 instead of the previous offset 0x10.

This offset can point to the previous PSP Directory header as defined in Table 3 on page 42; even if it only needs to support one program.

This offset can also point to the defined combo PSP directory table, in order to support multiple programs. For example, if it needs to support two program combos, Family 15h Models 70h-7Fh and Family 17h Models 00h-0Fh, offset 14 can first point to the combo PSP Directory table, then the two entries reside in the combo. The PSP Directory header will point to each programs PSP Directory table.

Similarly, starting from Family 17h Model 30h, the PSP off-chip bootloader will look up the BIOS directory pointer from the Embedded Firmware structure offset 0x28 instead of the previous program unique offset.

Table 10 describes the fields the PSP/BIOS Combo Directory header structure.

Table 10. PSP/BIOS Combo Directory Header

Field Name	Offset (Hex)	Size (In Bytes)	Description/Purpose
Cookie	0x00	4	Cookie “2PSP” (0x50535032) or “2BHD” (0x44484232) to recognize the header.
Checksum	0x04	4	32-bit CRC value of the header items below this field and the including all entries. Fletcher's checksum algorithm is used for CRC calculation.
Total Entries	0x08	4	Number of PSP Directory Table entries in the table.
Look Up Mode	0x0C	4	0 - Dynamic look up through all entries, 1 - PSP/chip ID match. Not used for BIOS Combo directory.
Reserved	0x10	16	Reserved - Set to zero.

Table 11 describe the Entry field that is followed by the PSP Combo Directory header.

Table 11. PSP Combo Directory Entry Fields

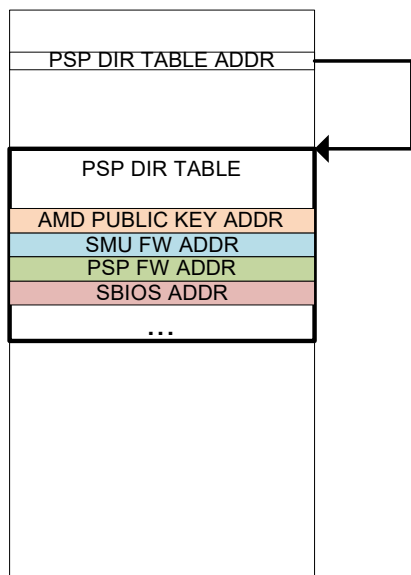
Field Name	Offset (Hex)	Size (In Bytes)	Description/Purpose
ID Select	0x00	4	0 - Compare PSP ID, 1 - Compare chip family ID
ID	0x04	4	32-bit Chip/PSP ID
PSP Directory Table (level 2) Address	0x08	8	Additional directory information (level 2). See Table 4.

The PSP Combo Directory header is used by PSP on-chip boot loader to distinguish the one level or two levels PSP directory structure. If it is one level, the PSP on-chip boot loader will load the PSP directory table as shown in the left part of Figure 4. If it is the two level structure, PSP on-chip boot loader will read the first level combo table, to get the level 2 table address, then it will look up the level 2 table as specified by LookUpMode.

If LookUpMode is 0, boot code will fetch the directory table for every entry, read AMD public root key, generate key hash and compare it with the value embedded in ROM until the hash check passes. If valid key is not found, the boot code will output an error message and stall the boot.

If LookUpMode is 1, boot code will read the ID from each combo BIOS entry and compare it with the value in silicon. Based on ID, Select field PSP or CHIP ID will be compared.

Without Combo Directory



With Combo Directory

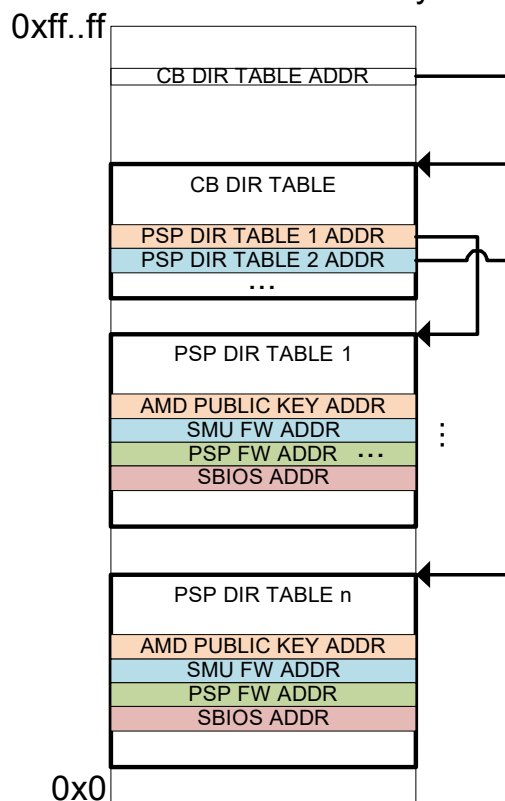
**Figure 4. PSP Directory, With or Without Combo Directory**

Table 12 lists valid PSP IDs per program.

Table 12. Valid PSP IDs per Program

Field Name	Value (Hex)
Family 17h Models 00h-0Fh	0xBC0900xx <i>Note: XX can be replaced with any value from 0x00 to 0xFF.</i>
Family 17h Models 10h-1Fh	0xBC0A00XX
Family 17h Models 20h-2Fh	0xBC0A01XX
Family 17h Models 30h-3Fh	0xBC0B00XX
Family 17h Models 60h-6Fh	0xBC0C00XX
Family 17h Models 70h-7Fh	0xBC0B0500
Family 19h Models 20h-2Fh	0xBC0B01XX
Family 19h Models 00h-0Fh	0xBC0B0DXX

Table 12. Valid PSP IDs per Program (Continued)

Field Name	Value (Hex)
Family 19h Models 10h-1Fh	0xBC0C0111
Family 19h Model 30h-3Fh	0xBC0B0Fxx
Family 19h Models 40h-4Fh	0xBC0D02xx

Note: Refer to B.1.4 “Node <COMBO_DIR>” on page 133 about how to use the “BuildPspDirectory” tool to create a combo directory structure.

4.1.4.3 One Level PSP Directory Layout

One level PSP directory layout means everything is in the PSP L1 directory and there is no PSP L2 directory. Previously, this layout was supported on each AMD program, but starting from Family 17h Models 60h-67h, one level PSP directory layout is not supported. Two levels of PSP directories become a requirement for both legacy BIOS and A/B BIOS. PSP Entry 0x0 (AMD public key) must be the first entry of the PSP L1 Directory, that is the PSP Boot ROM search sequence.

Table 13 shows the minimum entries needed for legacy BIOS in PSP L1:

Table 13. Legacy BIOS in PSP L1 Minimum Entries

Name	Type (Hex)	Description
AMD Public Key	0x00	AMD Public Key
PSP Boot Loader	0x01	PSP Stage 1 Boot Loader
Wrapped iKEK	0x21	Wrapped iKEK for the SoC
L2 PSP Directory	0x40	PSP L2 directory

Table 14 shows the minimum entries needed for A/B BIOS in PSP L1:

Table 14. A/B BIOS in PSP L1 Minimum Entries

Name	Type (Hex)	Description
AMD Public Key	0x00	AMD Public Key
PSP Boot Loader	0x01	PSP Stage 1 Boot Loader
Wrapped iKEK	0x21	Wrapped iKEK for the SoC
L2A PSP Directory	0x48	PSP L2 directory for partition A

4.1.5 BIOS Directory Table (AMD Family 17h and 19h Processor)

Some specific BIOS firmware related entries are removed from PSP Directory to comprise a defined BIOS Directory table.

BIOS Directory table provides information about various offchip BIOS components. This information is used by PSP firmware to locate and perform specific actions for each BIOS component type.

The BIOS Directory table structure is slightly different from PSP Directory as shown below.

- Multiple instances of firmware components are allowed for one specific type;
- The type field is further structured to reflect attributes of BIOS components such as "Region Type", "Reset Image", "Copy Image", "Read Only", that allows design flexibility for various purposes;
- The "Destination Address" field is added for specific entries that are expected to be copied from boot media to specific memory location.

Table 15 describes the fields of the BIOS Directory Table header structure.

Table 15. BIOS Directory Table Header Fields

Field Name	Offset (Hex)	Size (In Bytes)	Description/Purpose
BIOS Cookie	0x00	4	BIOS cookie "\$BHD" to recognize the header.
Checksum	0x04	4	32 bit CRC value of the header items below this field and the including all entries. Fletcher's checksum algorithm is used for CRC calculation.
Total Entries	0x08	4	Number of BIOS Directory Table Entries in the table.
Additional Info	0x0C	4	Additional directory information. See Table 4.

Table 16 describes the fields of BIOS Directory Table Entry structure.

Table 16. BIOS Directory Table Entry Fields

Field name	Offset (Hex)	Size (In Bits)	Description/Purpose
Type	0x00	8	Type encoding of BIOS entry. See Table 17, "BIOS Directory Table Entries" on page 57 for various encoding.
Region Type	0x01	8	Setup the memory region's security attribute for the BIOS entry
Reset Image	0x02	1	Boolean value to define the BIOS entry is a reset binary image
Copy Image	0x02	1	Define the binary image of the BIOS entry is for copying over to the memory region
Read Only	0x02	1	Setup the memory region for the BIOS entry to read only
Compressed (zlib)	0x02	1	Compressed using zlib

Table 16. BIOS Directory Table Entry Fields (Continued)

Field name	Offset (Hex)	Size (In Bits)	Description/Purpose
Instance	0x02	4	Specify the Instance of an entry
Subprogram	0x03	3	Specify the SubProgram, reference Section 4.1.6 for details
RomId	0x03	2	0: Content placed in the SPI device select by SPI_CS1_L. 1: Content placed in the SPI device select by SPI_CS2_L. For details on how to declare an entry with ROMID, refer to B.1.2, Node <PSP_DIR>.
Writable	0x03	1	1: Region writable 0: Region read only. This field must be filled when the Rom Armor feature is enabled.
Reserved	0x03	2	Reserved - Set to zero
Size	0x04	32	Memory Region Size : The window size of memory region AND if the "Copy Image" bit above is set. Size of the BIOS entry in bytes to be copied AND if the "Compressed(zlib)" flag is set, reflect the actual uncompressed size of the image, the 256 bytes of standard BIOS binary header is excluded, refer to "BIOS Directory Table (AMD Family 17h and 19h Processor)" on page 54 and "BIOS Boot x86 Initialization" on page 70 for more details.
Source Address	0x08	62	Depending on the Boot Media, Location: Physical Address* of SPI-ROM location where the data for the corresponding Entry is located.*Change to Offset of SPI ROM, start from AMD Family 17h Model 30h
Entry Address Mode	0x08	2	Same as Entry Address Mode in PSP directory table entry fields
Destination Address	0x10	64	Memory Region Location: Starting Address of memory region AND If the "Copy Image" bit above is set. Location: Destination Address of memory location where the data for the corresponding BIOS Entry is copied

4.1.5.1 AMD Processor BIOS Directory Entry Type Encodings

The BIOS directory table is a structure that can grow as new entries are identified. Table 17 defines the BIOS directory table entry types.

Table 17. BIOS Directory Table Entries

Entry Type Value	Description/Purpose
0x05	Location field points to the public part of the OEM/IBV BIOS Signing Key Token as defined in Appendix B
0x07	Location field points to the signature computed over the BIOS RTM code concatenated with BIOS Directory Table with the private part of BIOS Signing Key following the RSASSA-PSS signature scheme with SHA-256 used as the hashing algorithm for both message and mask generation in the little-endian format.
0x60	Location field points to the AGESA PSP Customization Block (APCB) data.
0x61	Location field points to the AGESA PSP Output Block (APOB) data. Declare as POINT_ENTRY in BuildPspDirectory tool XML file.
0x62	Location field points to BIOS binary images. We recommend this entry to be a POINT_ENTRY to avoid PSB signing issues.
0x63	Location field point to AGESA PSP Output Block (APOB) data NV copy, this data is referenced by PSP AGESA boot loader during S3 resume. AGESA will save the full copy of APOB (entry 0x61) to SPI by calling platform BIOS flash access library. In combo BIOS implementation, this region can be allocated to share between different program to save SPI footprint.
0x64	Location field pointing to the instruction portion of PMU firmware, for which the subtype is encoded per Table 1.
0x65	Location field pointing to the data portion of PMU firmware, for which the subtype is encoded per Table 1.
0x66	Location field pointing to the microcode patch
0x67	Core machine check exception data
0x68	Location field points to the backup copy of AGESA PSP Customization Block (APCB) data
0x69	Location field pointing to the interpreter binary that displays the video image.
0x6A	Location field pointing to the MP2 FW configuration file.
0x6B	Location field pointing to main memory, which is shared with x86 from PSP FW (BIOS/ Coreboot). This entry will be used in the context of ChromeOS, where vboot (running on PSP) will share workbuffer to Coreboot (running on x86).
0x6C	MPM specific Configurations which will be used by AMF
0x6D	RomArmor2, Location field points to BIOS variable flash NV store. This field is used in RomArmor2 to inform PSP of BIOS variable flash NV store region to be writable.

Table 17. BIOS Directory Table Entries (Continued)

Entry Type Value	Description/Purpose
0x6E	Debug unit, Deployment of various IP Debug Unit (DBGU) and Debug State Machine (DSM) setups.
0x70	Pointer to BIOS level 2 directory. Declare as POINT_ENTRY in BuildPspDirectory tool XML file.

Table 18. PMU Firmware Subtype Encoding

Sub Type Value	Description/Purpose
0x01	PMU firmware for DDR4 UDIMM 1D training.
0x02	PMU firmware for DDR4 RDIMM 1D training.
0x03	PMU firmware for DDR4 LRDIMM 1D training.
0x04	PMU firmware for DDR4 2D training.
0x05	PMU diagnostic firmware for DDR4 2D training

4.1.5.2 Using Compressed BIOS Image

Memory initialization is done before main cores are released. PSP supports the CCP4.0 Zlib engine. PSP firmware can utilize CCP engine to further decompress BIOS as part of the BIOS copy operation to DRAM. This design provides the feasibility to further compress BIOS reset binaries (e.g., PEI FV, to reduce the BIOS footprint size on the boot media). From experiments, both SPI-ROM size and PSP boot latency are significant reduced. Currently, PSP FW only supports zlib compressed format.

A few steps need to be added in the BIOS build process:

1. Compress the built BIOS component (e.g., PEI FV) using a zlib tool.
2. Add standard BIOS binary header to the compressed data, refer to 4.1.5.1 on page 57 for details.
3. Embed the compressed data in the final FD file
4. Update the PSPDirectory entry for respective BIOS components
 - a. Set "Compressed" flag to 1 of the BIOS directory entry in the xml configure file, please refer to Appendix B, Section B.1, "PSP Directory Configure File Format" on page 125.
 - b. Set source "Address" to point to the image in step 3 above. Note the "size" of entry should be the uncompressed size of the image.

During the cold boot, when PSP firmware finds a BIOS directory entry with the "compressed" flag set, it will locate the compressed image and decompress the image to the destination address based on the compressed image header, compare decompressed image size again to the size in the PSP entry, and Error stop if size is not matched.

After the decompress step, it will proceed with normal authentication and post steps.

4.1.5.3 Standard BIOS Binary Header

The standard PSP FW binary image header contains 256 bytes. It fills all fields with 0 except the following.

Table 19. Standard BIOS Binary Header

Offset	Size in Bytes	Contents
0x14	4	Compressed image size excluding the header.

4.1.6 SubProgram Field Within a Chip Family

Within a chip family all FW elements need to be put into one PSP Directory or BIOS Directory. However, some firmware types may require different modules for different chip variants within a chip family. To support this, a “subprogram” field has been specified to allow firmware to reference – in those cases where it matters – a specific subprogram variant of the desired firmware type.

"SubProgram" only applies to limited types of FWs, SMU FW (Type 0x8), Secure Policy Binary (Type 0x24), MP2 FW (Type 25) in PSP Directory, PMU FW, and APCB binaries in BIOS directory; for all other FW types this field should be 0. Reference Table 5. PSP Directory Table Entry Fields and Table 15. BIOS Directory Table Entry Fields for details offset.

Table 20 shows the encodings of the Sub-Program field for these limited types of FW. And this sub program value needs to be declared specifically in the PSP directory configuration XML file. Reference sample XML file in the AGESA PI for details.

Table 20. Sub-Programming Encoding

Sub-Program		Value
AMD Family 17h Models 00h-0Fh Processors	Models 00h-01Fh	0
	Models 08h Package AM4	1
	Model 08h Package SP3r2	2
AMD Family 17h Models 10h-2Fh Processors	Model 11h	0
	Models 20h-2Fh	1
	Model 18h	2
Family 17h Models 70h-7Fh (Client)		0
Family 19h Models 20h-2Fh (Client)		1
Family 17h Models 30h-3Fh (Server)		1
Family 19h Models 00h-0Fh (Server)		1
Family 19h	Models 10h-1Fh	0
	Models 40h-4Fh Rev A0	0
	Models 40h-4Fh Rev B0	1

4.1.7 EFS Search Algorithm

4.1.7.1 Client and Server Products in Family 17h (Starting from Models 30h-3Fh)

Starting with Family 17h Models 30h-3Fh, the SOC can support an SPI chip larger than 16MB, where multiple changes are required to the EFS and PSP/BIOS directory tables.

Changes in EFS Structure and PSP/BIOS Directory Table:

- New bit definition at EFS offset 0x24, named `second_gen_efs` (EFS_0x24[0])

Bit value 1: This is a first-generation EFS structure. It is ignored by PSP boot ROM.

Bit value 0: This is a second generation EFS structure.

The Boot ROM ignores the EFS that sets this bit to 1 and continues to search the EFS in another region, which allows to have two EFS in bottom 16M and upper 16M. This ROM layout can be used for multiple programs to create dual BIOS support. The previous program implementation is placed in the lower 16M, and the new program implementation is placed in the upper 16M.

- The address used in EFS and PSP/BIOS directory tables changed from the X86 physical MMIO address to the relative offset.

Before describing the implementation details for supporting multiple flash layouts, consider the fuse bits available to assist the EFS search algorithm, shown in Table 21.

Table 21. Fuse Bits to Assist EFS Search Algorithm

Sr. No.	Fuse Name	Description
1	FUSE_EFS_32M_TOP_HALF_SECOND	After checking the first 16MB of SMN for EFS, swap the first and second 16MB of SMN (with <code>Addr32_Ctrl3[SPI_ROM_page[0]]</code> (<code>SPIx5C[0]</code>) aka <code>XOR[24]</code>). Then, check for EFS a second time.
2	FUSE_2ND_GEN_EFS	Second Generation EFS is enabled. This means, in addition to checking for the 55AA55AA signature in the EFS, ensure that the "second_gen_efs (0x24[0])" bit in the EFS is clear.

Bootcode is always required to find the EFS at an offset in the first 16MB of SMN space (SMN 44FF_FFFF:4400_0000). To achieve this, it uses XORs of address bits 24 and 25 to remap the four 16MB SMN regions to the first 16MB region.

For LPC boot, Bootcode must set a new FCH bit `ROM2_Addr_override[force_spi_lpc_3124_1]` to force the upper 8 address bits high on LPC. LPC does not support BIOS images greater than 16MB.

Pseudo codes describing the BOOTROM search algorithm:

```

define check_efs(offset)
    return((read_location(SMN_base+offset)==0x55AA55AA) ? true : false)
enddefine

define check_2nd_gen_efs(offset)
    return((read_location(SMN_base+offset+0x24) & 0x00000001) ? false : true)
enddefine

```

```

define adjust_lpc_addr()
    if lpc_boot_strap_is_set()
        set_ROM2_Addr_override_force_spi_lpc_3124_1(1) # new FCH bit
    end
enddefine

define check_efs_offsets()
    for offset in FA_0000 F2_0000 E2_0000 C2_0000 82_0000 02_0000 do
        if check_efs(offset)
            if (fuse_is_clear(FUSE_2ND_GEN_EFS) || check_2nd_gen_efs(offset))
                return true
            end
        end
    done
    return false
enddefine

// Same boot code sequence covers all cases and always finds EFS in the bottom
16MB of SMN address space.

adjust_lpc_addr()
if alt_image_strap_is_set()                                # strap not set
    set_spi_xor_bit25(1)
end
if not check_efs_offsets()                                # EFS found (bottom 16MB of SMN)
    if fuse_is_set(FUSE_EFS_32M_TOP_HALF_SECOND)
        set_spi_xor_bit24(1)
        if not check_efs_offsets()
            hang_known_post_code(NO_EFS_FOUND)
        end
    end
    hang_known_post_code(NO_EFS_FOUND)
end

```

Multiple SPIROM flash layouts are supported:

- **Single 16M in 16MB SPIROM (1 X 16)**

- One EFS can be placed in binary offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000, and 02_0000, with priority from high to low.
- Set EFS offset 0x24 bit second_gen_efs (EFS_0 x 24[0]) to 0.
- All addresses must change from X86 MMIO physical address to relative offset in 16MB BIOS image.

For EFS, it is required to change by EFS creator manually.

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Use the whole 16M binary as input to "BuildPspDirectory" tool.

- **Single 16M in 32MB SPIROM (1 X 32)**

- One EFS can be placed at low 16M region, at offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000 and 02_0000, with priority from high to low.
- Set EFS offset 0x24 bit second_gen_efs (EFS_0 x 24[0]) to 0.
- All addresses must change from X86 MMIO physical address to relative offsets in 32M BIOS image.

For EFS, it is required to change by EFS creator manually,

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Use the whole 32M binary as input to "BuildPspDirectory" tool.

- **Two 16M in 32MB SPIROM (2 X 16)**

- Two EFS placed separately at bottom 16M and top 16M regions.
- Set EFS offset 0 x 24 bit second_gen_efs (EFS_0 x 24[0]) to 1 in bottom 16M for old programs.
- Set EFS offset 0 x 24 bit second_gen_efs (EFS_0 x 24[0]) to 0 in top 16M for new programs in Family 17h.

Only one EFS entry is allowed with EFS offset 0 x 24 bit second_gen_efs (EFS_0 x 24[0]) to 0 in one BIOS binary.

- EFS can be placed in the 16M region with binary offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000 and 02_0000, with priority from high to low.
- All addresses must change from X86 MMIO physical address to relative offsets in 16M BIOS image.

For EFS, it is required to change by EFS creator manually.

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Call "BuildPspDirectory" twice separately for bottom 16M and upper 16M; copy or concatenate them together to create final BIOS image.

4.1.7.2 Server Products in Family 19h (Starting from Model 00h-0Fh)

Starting with Family 19h Models 00h-0Fh, the SOC extends the SPIRROM support to Multi-Gen EFS. Changes are required to the EFS and PSP/BIOS directory tables.

Changes to EFS Structure and PSP/BIOS Directory Table:

- New bit definition at EFS offset 0x24, multi_gen_efs[15:0] (EFS_0x24[15:0]).

A bit that is clear in this 16-bit field indicates that the EFS is compatible with generations of processors checking that bit. Multiple bits may be clear to indicate EFS compatibility with multiple processor generations.

Bit value 1: This is a first-generation EFS structure. It is ignored by PSP Boot ROM.

Bit value 0: This EFS is compatible and valid for FUSE_MULTI_GEN_EFS_VALUE[3:0].

With Multi-Generation EFS, two EFS can be integrated into bottom 16M and upper 16M regions. This layout can be used to create dual BIOS support with multiple programs, where the previous program implementation is placed in the lower 16M, and the new program implementation is placed in the upper 16M (and vice versa).

- The address used in EFS and PSP/BIOS directory tables changed from the X86 physical MMIO address to the relative offset.

Before describing the implementation details for supporting multiple flash layouts, consider the fuse bits available to assist the EFS search algorithm, shown in Table 22.

Table 22. Fuse Bits to Assist EFS Search Algorithm

Sr. No.	Fuse Name	Description
1	FUSE_MULTI_GEN_EFS_EN	Multi-Generation EFS search enabled. When set, in addition to checking for the 55AA55AA signature in the EFS, Bootcode checks that the bit of EFS field multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) is zero and considers this a valid EFS.
2	FUSE_MULTI_GEN_EFS_VALUE	Value of Multi-Generation EFS
3	FUSE_EFS_32M_TOP_16M_FIRST	1: Before checking for EFS, swap the second 16MB of SPI into the first 16MB of SMN by setting Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]) to 1. 0: Do not change Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]) before checking for EFS.
4	FUSE_EFS_32M_CHECK_OTHER_16M	1: After checking for EFS, if EFS is not found, swap the first and second 16MB of SMN by inverting Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]). Then, check for EFS a second time. 0: After checking for EFS, if EFS is not found, do not check for EFS a second time.

Bootcode is always required to find the EFS at an offset in the first 16MB of SMN space (SMN 44FF_FFFF:4400_0000). To achieve this, it uses XORs of address bits 24 and 25 to remap the four 16MB SMN regions to first 16MB region.

For LPC boot, Bootcode must set a new FCH bit ROM2_Addr_override[force_spi_lpc_3124_1] to force the upper 8 address bits high on LPC. LPC does not support BIOS images greater than 16MB.

Pseudo codes describing the BOOTROM search algorithm:

```
define check_efs(offset)
    return((read_location(SMN_base+offset)==0x55AA55AA) ? true : false)
enddefine

define check_multi_gen_efs(offset)
    return((read_location(SMN_base+offset+0x24) & (0x00000001 <<
FUSE_MULTI_GEN_EFS_VALUE)) ? false : true)
enddefine

define adjust_lpc_addr()
    if lpc_boot_strap_is_set()
        set_ROM2_Addr_override_force_spi_lpc_3124_1(1) # new FCH bit
    end
enddefine

define check_efs_offsets()
    for offset in FA_0000 F2_0000 E2_0000 C2_0000 82_0000 02_0000 do
        if check_efs(offset)
            if (fuse_is_clear(FUSE_MULTI_GEN_EFS_EN) || check_multi_gen_efs(offset))
                return true
            end
        end
    done
    return false
enddefine

// Same boot code sequence covers all cases and always finds EFS in the bottom
16MB of SMN address space.
adjust_lpc_addr()
if alt_image_strap_is_set() # strap not set
    set_spi_xor_bit25(1)
end
if fuse_is_set(FUSE_EFS_TOP_16M_FIRST)
    set_spi_xor_bit24(1)
```



```

end
if not check_efs_offsets()                                # EFS found (bottom 16MB of SMN)
  if fuse_is_set(FUSE_EFS_CHECK_SECOND_16M)
    set_spi_xor_bit24(not get_spi_xor_bit24())
    if not check_efs_offsets()
      hang_known_post_code(NO_EFS_FOUND)
    end
  end
end
hang_known_post_code(NO_EFS_FOUND)
end

```

Multiple SPIROM flash layouts are supported:

- **Single 16M in 16MB SPIROM (1 X 16)**

- One EFS can be placed in binary offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000, and 02_0000, with priority from high to low.
- Set EFS offset 0x24 bit multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) to 0.
- All addresses must change from X86 MMIO physical address to relative offset in 16MB BIOS image.

For EFS, it is required to change by EFS creator manually.

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Use the whole 16M binary as input to the "BuildPspDirectory" tool.

- **Single 32M in 32MB SPIROM (1 X 32)**

- One EFS can be placed at low 16M region, at offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000 and 02_0000, with priority from high to low.
- Set EFS offset 0x24 bit multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) to 0.
- All addresses must change from X86 MMIO physical address to relative offsets in 32M BIOS image.

For EFS it required to change by EFS creator manually,

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Use the whole 32M binary as input to "BuildPspDirectory" tool.

- **Two 16M in 32MB SPIROM (2 X 16)**

- Two EFS placed separately at bottom 16M and top 16M regions for different products/programs.
- The assumption here is that the FUSE_MULTI_GEN_EFS_VALUE[3:0] value for both programs/products are different, and respective BIOS images have EFS offset 0x24 bit

multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]]
(EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) set to 0.

- EFS can be placed in the 16M region with binary offset FA_0000, F2_0000, E2_0000, C2_0000, 82_0000 and 02_0000, with priority from high to low.
- All addresses must change from X86 MMIO physical address to relative offsets in 16M BIOS image.

For EFS, it is required to change by EFS creator manually.

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Call "BuildPspDirectory" twice separately for bottom 16M and upper 16M; copy or concatenate them together as the final BIOS image.

4.1.7.3 Products in Server Family 19h Model 10h-1Fh

Family 19h Models 10h-1Fh include changes in how the EFS search algorithm is implemented with the Multi-Gen EFS support described above. As a result, changes are required to the EFS and PSP/BIOS directory tables.

- Changes to EFS structure and PSP/BIOS Directory Table:

- Only one EFS offset @0x0002_0000 is supported.
- New bit definition in EFS offset 0x24, multi_gen_efs[15:0] (EFS_0x24[15:0]).

A bit that is clear in this 16-bit field indicates that the EFS is compatible with generations of processors checking that bit. Multiple bits may be clear to indicate EFS compatibility with multiple processor generations.

Bit value 1: This is a first-generation EFS structure. It is ignored by PSP Boot ROM.

Bit value 0: This EFS is compatible and valid for FUSE_MULTI_GEN_EFS_VALUE[3:0].

With Multi-Generation EFS, a maximum of four EFS can be integrated in 64MB SPIROM Flash.

- The address used in EFS and PSP/BIOS directory tables changed from the X86 physical MMIO address to the relative offset.
- Server programs starting from Family 19h Model 10h-1Fh only support 16MB BIOS image, per product/program.

Before describing the implementation details for supporting multiple flash layouts, consider fuse bits available to assist the EFS search algorithm, shown in Table 23.

Table 23. Fuse Bits to Assist EFS Search Algorithm

Sr. No.	Fuse Name	Description
1	FUSE_MULTI_GEN_EFS_EN	Multi-Generation EFS search enabled. When set, in addition to checking for the 55AA55AA signature in the EFS, Bootcode checks that the bit of EFS field multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) is zero and considers this a valid EFS.
2	FUSE_MULTI_GEN_EFS_VALUE	Value of Multi-Generation EFS
3	FUSE_EFS_32M_TOP_16M_FIRST	1: Before checking for EFS, swap the second 16MB of SPI into the first 16MB of SMN by setting Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]) to 1. 0: Do not change Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]) before checking for EFS.
4	FUSE_EFS_32M_CHECK_OTHER_16M	1: After checking for EFS, if EFS is not found, swap the first and second 16MB of SMN by inverting Addr32_Ctrl3[SPI_ROM_page[0]] (SPIx5C[0] aka XOR[24]). Then, check for EFS a second time. 0: After checking for EFS, if EFS is not found, do not check for EFS a second time.
5	FUSE_TOP_SMN_BOOT_APERTURE	1: Boot from SMN aperture 47FF_FFFF:4700_0000 0: Boot from SMN aperture 44FF_FFFF:4400_0000
6	FUSE_EFS_CHECK_3RD4TH_16M	1: Search all 4 regions of 16Mbytes of SPIROM Flash (total 64MB). 0: Search only 2 regions (total 32MB).

Bootcode is always required to find the EFS at an offset in the fourth 16MB of SMN space (SMN 47FF_FFFF:4700_0000).

Pseudo codes describing the BOOTROM search algorithm:

```
# change SMN_base here to 4700_0000
define check_efs(offset)
    return((read_location(SMN_base+offset)==0x55AA55AA) ? true : false)
enddefine

define check_multi_gen_efs(offset)
    return((read_location(SMN_base+offset+0x24) & (0x00000001 <<
FUSE_MULTI_GEN_EFS_VALUE)) ? false : true)
enddefine
```

```
define check_efs_offsets()
    // Only supported offset is 02_0000
    if check_efs(offset)
        if (fuse_is_clear(FUSE_MULTI_GEN_EFS_EN) || check_multi_gen_efs(offset))
            return true
        end
    end
done
return false
enddefine

// Same boot code sequence covers all cases and always ends up finding EFS in the
// top 16MB of SMN address space.
if alt_image_strap_is_set()
    set_spi_xor_bit25(1) # no harm in eSPI/LPC
end
if fuse_is_not_is_set(FUSE_EFS_TOP_16M_FIRST)
    set_spi_xor_bit24(1) # no harm in eSPI/LPC
end

if not check_efs_offsets()
    # eSPI/LPC need not check further but no harm checking
    if fuse_is_set(FUSE_EFS_CHECK_SECOND_16M)
        set_spi_xor_bit24(not get_spi_xor_bit24())
        if not check_efs_offsets()
            if fuse_is_set(FUSE_EFS_CHECK_3RD4TH_16M)
                set_spi_xor_bit24(not get_spi_xor_bit24())
                set_spi_xor_bit25(not get_spi_xor_bit25())
                if not check_efs_offsets()
                    set_spi_xor_bit24(not get_spi_xor_bit24())
                    if not check_efs_offsets()
                        hang_known_post_code(NO_EFS_FOUND)
                    end
                end
            end
        else
            hang_known_post_code(NO_EFS_FOUND)
        end
    end
end
```

```

        end
    end
end
hang_known_post_code(NO_EFS_FOUND)
end

```

Multiple SPIROM flash layouts are supported:

- **Single 16M (1 X 16)**

- Place EFS at binary offset 02_0000.
- Set EFS offset 0x24 bit multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) to 0.
- All addresses must change from X86 MMIO physical address to relative offset in 16MB BIOS image.
For EFS, it is required to change by EFS creator manually,
For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">
- Use the whole 16M binary as input to the "BuildPspDirectory" tool.

- **Two 16M in 32MB SPIROM (2 X 16)**

- Two EFS placed separately at bottom 16M and top 16M regions, for different products/programs.
- The assumption here is that the FUSE_MULTI_GEN_EFS_VALUE[3:0] value for both programs/products are different, and respective BIOS images have EFS offset 0x24 bit multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]] (EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) set to 0.
- Place EFS in the 16M region at binary offset 02_0000.
- All addresses must change from X86 MMIO physical address to relative offsets in 16M BIOS image.
For EFS, it is required to change by EFS creator manually.
For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">
- Call "BuildPspDirectory" twice separately for bottom 16M and upper 16M; copy or concatenate them together to create final BIOS image.

- **Four 16M in 64MB SPIROM (4 X 16)**

- Four EFS placed separately at all four 16M regions, for different products/programs.
- The assumption here is that the FUSE_MULTI_GEN_EFS_VALUE[3:0] value for all the programs/products are different, and respective BIOS images have EFS offset 0x24 bit

multi_gen_efs[FUSE_MULTI_GEN_EFS_VALUE[3:0]]
(EFS_0x24[FUSE_MULTI_GEN_EFS_VALUE[3:0]]) set to 0.

- Place EFS in the 16M region at binary offset 02_0000.
- All addresses must change from X86 MMIO physical address to relative offsets in 16M BIOS image.

For EFS, it is required to change by EFS creator manually.

For PSP/BIOS directory, set "Addressmode" in the XML file to 1: <DIRS AddressMode="1">

- Call "BuildPspDirectory" four times separately for all 16M regions; copy or concatenate them together to create final BIOS image.

4.2 Runtime Execution Flow

The high level description of execution flow is listed below.

4.2.1 BIOS Boot x86 Initialization

After the x86 cores are released from reset, the BIOS boot phase starts and performs the following steps:

1. The CPU fetches the BIOS reset vector which filled by BIOS Directory type 0x62, for more information please refer to Section 4.4.
2. The BIOS SEC code loads and executes rest of PEI driver in PEI (RTM) volume that has already been authenticated by PSP firmware.
3. If the platform wishes to use a TPM, the platform may use firmware TPM services offered by PSP or it may use discrete TPM, but not both; the correct BIOS TPM drivers need to be present to perform necessary TPM operation. PSP firmware offers TPM 2.0 support as outlined in Microsoft specification. BIOS must wait for memory to be available before sending any command to PSP.
4. For the program which memory initialized by BIOS, BIOS sends mailbox command "MboxBiosCmdDramInfo" to PSP to inform availability of memory once memory is initialized. PSP firmware can now use memory for its own use.
5. At the end of PEI stage the PEI kernel must authenticate DXE volume before handing off control to DXE IPL. As part of the build process, the digest of DXE volume is saved within PEI volume. PEI core computes the digest of DXE volume and compares it against saved digest in PEI volume. If the digest matches, the DXE IPL is considered trusted and PEI kernel code continues to load the DXE driver.
6. At this point the BIOS boot process moves to the DXE phase.
7. The DXE core authenticates various DXE driver modules, such as Option ROMs, third party DXE drivers, etc. that are not part of the SPI flash (i.e., not part of DXE volume). The authentication steps follow secure boot flow as outlined in UEFI specification.

8. If the BIOS is using integrated TPM, then the fTPM drivers are loaded to perform measurements of other DXE components. The fTPM DXE driver allocates necessary memory space for TPM request/response buffer and builds the TPM2 Static ACPI table as outlined in the Microsoft whitepaper on "Trusted Execution Environment ACPI Profile". In addition it extends necessary protocol outlined in TCG specification to aid BIOS to perform measurement and other TPM use. A platform should only include discrete or integrated TPM and not both. The only exception is AMD reference board where both kind of TPM are present on board for validation purposes only. In such validation boards BIOS need to support both kind of TPM driver and provide setup option (default to internal TPM) for test user to select discrete vs. internal TPM device for test.
9. Next the SMM environment is built and SMM drivers are loaded. The SMM drivers are needed to handle PSP firmware storage request. PspP2Cmbox SMM driver: This driver handles SMI requests coming from PSP firmware and also provides information to PSP FW of how to trigger specific SMI to BIOS. The Fake SMI has been selected as the SMI source. Currently all the P2C mailbox commands are only used for providing the SPI ROM access to PSP FW. Once the SMM driver handler receives the request from PSP FW, it will call IBV/OEM customized storage library in the backend.
10. The customized storage library operates on SPI space for multiple PSP NVRAM region and enables write/erase operation on that space. It also ensures NVRAM region is appropriately locked against unauthorized update on this SPI space. Note, this library is required to work in SMM mode, and should not depend on any boot available services.
11. The PspP2Cmbox SMM driver will perform the following steps:
12. The driver allocates transfer buffer in SMRAM for PSP to pass the parameter.
13. The driver sends a mailbox command "MBoxBiosCmdSmmInfo" to the PSP firmware and informs the space reserved for PSP communicating as well as other relevant information for PSP to trigger SMI at runtime.
14. When PSP firmware triggers SMI for BIOS services, the callback handler checks the integrity of data that PSP firmware writes into SMM buffer and uses IBV/OEM customized storage library to service PSP NVRAM access request.
15. Now PSP firmware can call the BIOS services to store NV data.
16. A PSP DXE/SMM driver is loaded to handle the resume path. This driver prepares the BIOS resume code (discussed in Chapter 5, BIOS S3-Resume Path Handling) in memory and sends a mailbox command to inform the PSP firmware about the location of the BIOS S3 resume vector. In addition it registers to service the SlpxSx SMM trap. (See Chapter 5 on page 36 for more details on S3 resume path).
17. The IBV Storage code needs to lock the PSP and other critical regions of the SPI flash. The SPI region must be locked before running any Option ROMs or other non-System ROM code and only trusted SMM code should be able to unlock and update this region.
18. PSP DXE driver will register a callback on ReadyToBoot event to perform the tasks right before handling control to OS boot loader. The details steps of the callback are:

19. Initialize the RDRAND instruction related register
20. Save the SMM resume vector and Core context to specific MSR (Note, should be only called once during boot)
21. Send MboxBiosCmdBootDone C2P mailbox command to PSP FW
22. The remaining DXE drivers are loaded and finally OS boot loader is loaded and after proper authentication BIOS hands off control to the OS boot loader.

4.2.2 BIOS Runtime Functionality

The high-level execution flow during runtime is listed below.

After BIOS hands off control to the OS boot loader, the OS boot loader loads the operating system (e.g., Windows 8).

1. The OS takes control and loads rest of the OS drivers.
2. The OS references the fTPM ACPI table and loads fTPM OS driver to offer TPM support.
3. The OS uses the TPM software stack and sends TPM commands that are handled by PSP firmware. The PSP firmware services any OS requests for TPM commands.
4. If the PSP firmware needs to perform a write to SPI storage, it performs the following additional steps:
 - a. PSP firmware copies necessary data as well as command parameters into SMM space reserved for PSP-BIOS communication.
 - b. PSP firmware triggers SMI for BIOS to service the request. BIOS SMM entry code invokes all registered SMM handlers. In this case, the P2CMbox SMM Handler gets control and checks if the PSP firmware generated the SMM.
 - c. The P2CMbox SMM callback handler locates and invokes the IBV/OEM customized storage library to write the data into SPI.
 - d. Finally, the P2CMbox SMM driver reports the appropriate status to the PSP firmware and resumes out of SMM mode.
5. The PSP firmware returns the appropriate status and data values to the operating system.

4.3 Optimized Boot Flow

Starting with Family 17h Models 00h-0Fh Processors, the cache as RAM feature has been removed. To support this, it requires the processor's ability to run from DRAM after X86 processor release from reset.

The PSP FW copies BIOS FW from boot media to specific Dram Location before releasing the main core. BIOS Image needs to be updated to allow it to run from a fixed DRAM location that is applicable per platform design. An optimized boot path is proposed as below where BIOS code is fetched from DRAM during CPU reset.

High level boot flow view

1. PSP off-chip FW loads BIOS image from boot media to DRAM. The BIOS source and DRAM destination location will be passed via PSP Directory entries. (See implementation notes below). PSP firmware will load remaining PSP off-chip components (secure OS etc.) before releasing the core. Once the cores are released PSP firmware should not directly access boot media
2. On x86 systems, The standard CPU reset vector does not point to DRAM address; instead it points to MMIO 0xFFFFF0 address. To change this, PSP will hold the core in reset and setup BootSaveArea in C6 region for each core such that on core release CPU microcode will update CPU register context per BootSaveArea (like in Secure S3 resume). This BootSaveArea holds information of CPU register such as rip, rax,... gdt etc that reflect the state of CPU register at first fetch. Platform BIOS can set the reset vector through the BIOS binary images (type 0x62) is has "Reset Image" Bit set. (see step 6. b. for details)
3. Elements of PSP directory are signed; and PSP authenticates each component before copying those to DRAM. In addition PSP fw will ensure the regions don't overlap (to protect against hacking of PSP directory against overlay images). If the image is corrupted or tempered, PSP firmware reports this error, as it does for other error scenarios.
4. PSP firmware releases the core. CPU microcode (x86 only) updates the CPU registers value per BootSaveArea including the rip value and cpu core will fetch BIOS code from DRAM location that was copied in step (2). And reflected by rip updated value.
 - e. On x86, CPU will fetch from DRAM location per BootSaveArea and BIOS reset binary image's destination. The reset vector is calculated as:
 - BiosResetVector[Physical address] = BIOS image destination + BIOS image size - 0x10
 - BiosResetVector[Segment] = BIOS image destination + BIOS image size - 0x10000
 - BiosResetVector[Offset] = 0xFFFF0
5. Main core will fetch and execute BIOS from DRAM. On x86 system BIOS does not need to setup CAR etc and can use DRAM straight away. On x86 BIOS will run in 16bit real mode. The Dram Information can be retrieve from PSP APOB Structure. On ARM the BIOS will run in 64bit EL3 mode.
6. BIOS code will run through rest of the flow SEC, PEI, DXE and later handover control to OS
7. BIOS will mark this DRAM region as reserved for the OS. This region will be reported reserved via FDT/UEFI services etc. as supported by OS. The reserve region will include BIOS runtime services and BIOS PEI drivers needed during resume from sleep.

Alternatively, BIOS can use SMM area for post OS handling such that resume from sleep, etc. starts in SMM mode and perform SOC init and rsm to OS resume vector in SMM space. This is achieved by programming MSR C001_10E0 from SMM mode during boot; however migrates all resume code within SMM requires significant change in existing UEFI BIOS and not recommended at this point.

If a platform BIOS need support a processor doesn't support "Dram boot" and a processor support "Dram boot" in one single SPI binary. Platform can create two PEI FVs, one for Non-Dram boot, and one for Dram boot, these two PEI FVs need be relocated during the BIOS build, to make sure the PEI

FV can be run on the specific memory address. The PEI FV for "Non-Dram boot" can be still placed at bottom of binary file in order to support previous programs, which enforce the reset vector at 0xFFFFFFFF0. The PEI FV for "Dram boot" can be placed any location of SPI binary, just need to make sure this location declared accurately in the PSP Directory configure file. In the latest EFI implementation, the build tool can't support a FD with two PEI FVs. Alternately, multiple FD files can be declared in EFI build file, and append at the end of build

4.4 APCB Recovery

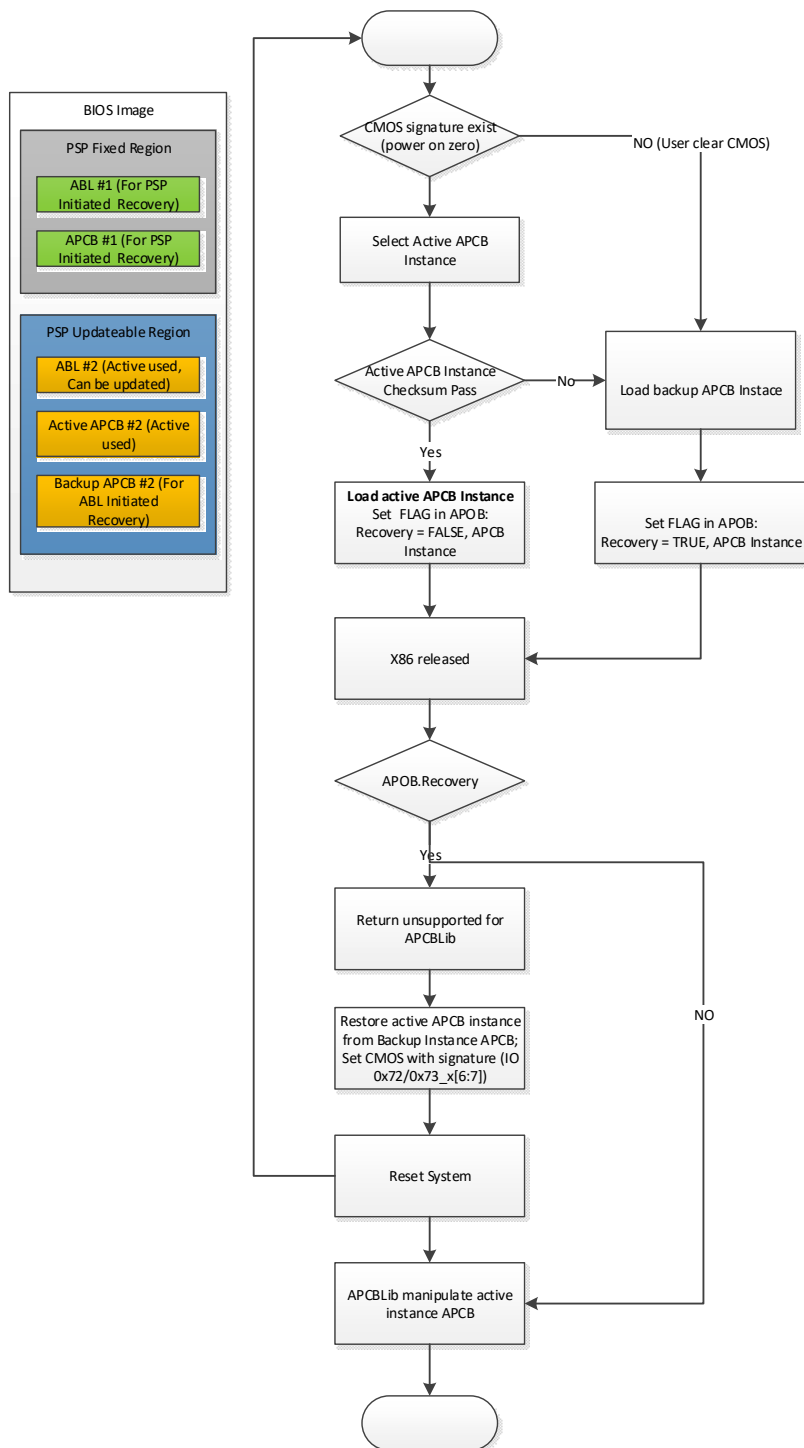
The APCB contains the key platform customization data for the ABL. The system will not boot if the APCB is corrupted or contains invalid data. So it requires a robust recovery mechanism to make sure the system can always boot even when the APCB is invalid and the corrupted or the invalid APCB can be recovered in certain cases.

This mechanism will cover two cases of an invalid APCB: 1) APCB data corrupted, it might be caused by incorrect SPI operation or unexpected power off during the APCB save 2) Invalid APCB setting lead the system can't boot especially when end-user set an aggressive value to the APCB setting.

The general ideas of APCB recovery are:

- Keep additional backup APCB safe copy to make sure system can always boot when active APCB can't work properly
- A mechanism to detect APCB integrity. The check sum field of APCB will be used for this purpose.
- A mechanism to pass the recovery flag to ABL phase to X86 phase. APOB with additional field added will be used for this purpose
- A mechanism to pass current active APCB instance. APOB with additional field added will be used for this purpose
- A mechanism to allow user to force the APCB manually, in case of set an invalid APCB setting. CMOS clear will be used as signal to force APCB recovery.

The whole flow for APCB recovery will be like the chart shown in Figure 5 on page 75:

**Figure 5. APCB Recovery Flow**

The detailed requirements for platform BIOS are as below:

1. Add APCB backup entry (0x68) instance for each active APCB active entry instance.
2. In multiple APCB case, the "BoardIdGettingMethod" should be identical for both APCB backup entry & active entry with same instance
3. The size of active APCB instance should be great than or equal to backup APCB instance
4. Reserve CMOS shadow region (IO_Index72/73[0x5:0x7]) for ABL CMOS power loss check.

4.5 PSP Initiated Crisis A/B Recovery Path

4.5.1 Overview

This section outlines the AMD design of PSP initialized Crisis recovery. This is an optional feature. Starting from Family 17h Models 10h-1Fh, it has been replaced with the A/B recovery scheme, which provides more robustness and scalability. The customer needs to take both the SPI space recovery binary occupied and combo BIOS support into consideration. As some regions can't be updated, this recovery feature has the limitation to support the case which requires forward compatibility. The primary focus here is to handle the case when firmware in PSP/BIOS directory region corrupted. The approaches that PSP take to attempt the recovery was to keep extra critical set of firmware in a fixed region, and PSP will load firmware from the fixed region in the case of authenticating firmware fails from the updatable region.

The goal of the firmware in the fixed region is to allow system boot to BIOS recovery path, and satisfies BIOS recovery mode basic requirement.

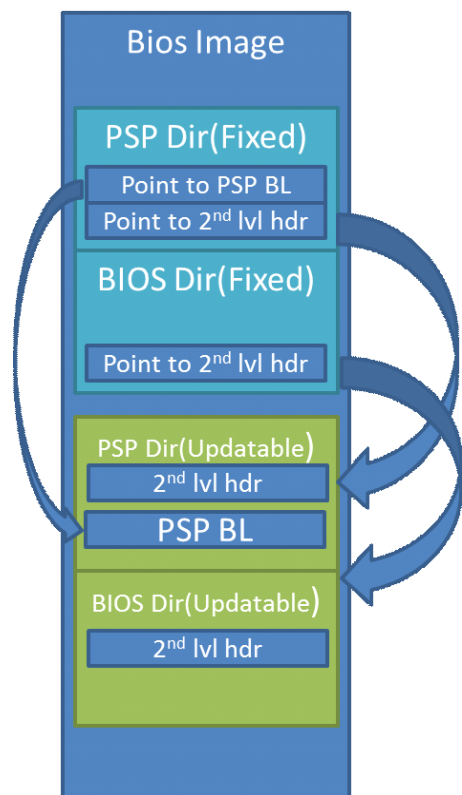
And below are the detail definitions for the two regions

- Fixed Region
The contents in this region are unchanged and not updatable during flash update, it is protected by the IBV or OEM flash tool.
- Updateable Region
This area can be updated by new releases containing bug fixes, enhancements, and new feature enablement code.

4.5.2 PSP/BIOS Directory Update

In order to create two separated PSP/BIOS Directory structure for both fixed and updatable region in one BIOS image file, 2 levels directory structure is used. All recovery critical and fixed firmware will be built in to the level 1 directory, the updatable firmware will be placed in the level 2 directory, and in the level 1 directory there is an entry used to point to the level 2 directory. Only exception is that PSP directory level 1 should contain an entry point to PSP off-chip boot loader, as immutable PSP on-chip ROM is not recognized the 2 level directory format. It also implies that PSP off-chip boot-loader should always be place at a fixed address, so it suggested to put the PSP off-chip boot-loader as 1st entry of PSP updatable region, due the fact the 1st entry is always place at offset 0x400.

Below is the diagram showing the layout of BIOS image with 2 level directories.

**Figure 6. Layout of BIOS Image with 2 Level Directories****4.5.2.1 Level 1 Firmware for AMD Family 17h Models 00h-0Fh Processors**

As describe above, Recovery critical firmware will be place in level 1 directory, below table list required firmware entries. Please reference sample file provided by AGESA PI for latest update.

Table 24. PSP Directory Level 1

Entry Type	Firmware Filename / Description
0x00	AmdPubKeyZP.bin
0x01	Pointing to the PSP Boot loader in PSP Level 2 Directory
0x03	PSP recovery bootloader
0x08	SmuFirmwareZP.csbin
0x0A	ZeppelinOemAbIw.tkn
0x21	Wrapped iKEK for the SoC
0x30	AgesaBootloader0_prod_ZP.csbin
0x31	AgesaBootloader1_prod_ZP.csbin

Table 24. PSP Directory Level 1(Continued)

Entry Type	Firmware Filename / Description
0x32	AgesaBootloader2_prod_ZP.csbin
0x33	AgesaBootloader3_prod_ZP.csbin
0x34	AgesaBootloader4_prod_ZP.csbin
0x35	AgesaBootloader5_prod_ZP.csbin
0x36	AgesaBootloader6_prod_ZP.csbin
0x24	RsmuSecurityPolicy_ZP.sbin
0x40	Point to PSP Directory level 2
0x05	BIOS public key signed by AMD private key
0x07	BIOS RTM + Level 1 Directory header signed by BIOS private key
0x60	APCB_ZP_D4.bin
0x68	Recovery APCB_ZP_D4.bin
0x61	Describe the address to hold AGESA PSP Output Block (APOB) data
0x62	Pointing to BIOS RTM images
0x64	Pointing to the instruction portion of PMU firmware,
0x65	Pointing to the data portion of PMU firmware
0x70	Pointing to BIOS Directory level 2

4.5.2.2 Level 2 Firmware for AMD Family 17h Models 00h-0Fh Processors

As describe above, the updatable firmware will be place in level 2 directory, below table list detail firmware entries. Please reference sample file provided by AGESA PI for latest update.

P

Table 25. PSP Directory Level 2

Entry Type	Firmware Filename / Description
0x01	PspBootLoader_prod_ZPSCM.sbin
0x02	PspSecureOs_prod_ZP.csbin
0x08	SmuFirmwareZP.csbin
0x09	ZeppelinSecureDebug.tkn
0x0A	ZeppelinOemAblFw.tkn
0x0B	PSP Soft fuse
0x0C	PspTrustlets_prod_ZP.cbin
0x0D	ZeppelinTrustlet.tkn
0x12	SmuFirmware2ZP.csbin
0x13	DebugUnlock_ZP.sbin
0x21	PspIkekZP.bin

Table 25. PSP Directory Level 2(Continued)

Entry Type	Firmware Filename / Description
0x22	SecureEmptyToken.bin
0x24	RsmuSecurityPolicy_ZP.sbin
0x30	AgesaBootloader0_prod_ZP.csbin
0x31	AgesaBootloader1_prod_ZP.csbin
0x32	AgesaBootloader2_prod_ZP.csbin
0x33	AgesaBootloader3_prod_ZP.csbin
0x34	AgesaBootloader4_prod_ZP.csbin
0x35	AgesaBootloader5_prod_ZP.csbin
0x36	AgesaBootloader6_prod_ZP.csbin

Table 26. BIOS Directory Level 2

Entry Type	Firmware Filename / Description
0x05	BIOS public key signed by AMD private key
0x07	BIOS RTM + Level 1 Directory header + Level 2 Directory header signed by BIOS private key
0x60	APCB binary
0x68	Recovery APCB binary
0x61	Describe the address to hold AGESA PSP Output Block (APOB) data
0x62	Pointing to BIOS RTM images
0x63	APOB non-volatile copy
0x64	Pointing to the instruction portion of PMU firmware
0x65	Pointing to the data portion of PMU firmware
0x66	Microcode patch file

4.5.3 Recovery Scenario for PSP Boot Loader

4.5.3.1 PSP On-Chip ROM Failed to Validate PSP BL

For the case if PSP on-chip Rom failed to validate and load the PSP production boot loader, PSP on-chip ROM should attempt to load PSP recovery from PSP directory entry 0x3 and execute the recovery path.

4.5.3.2 PSP Off-Chip BL Failed to Validate SMU/ABL/Security Gasket

For cases if PSP off chip boot loader failed to validate SMU firmware, AGESA Boot loader and security gasket policy file. PSP off chip boot loader enters the recovery mode and attempt to load the recovery entry for the failed firmware and subsequence firmware. After x86 is released, it notifies BIOS that PSP required the system to be in recovery mode.

4.5.4 Platform Secure Boot Related Changes

In 2 level BIOS directory structure, the below update needs to be applied:

1. Add Entry 5 (BIOS public key signed by AMD private key) for both level 1 and level 2
2. Fill the "BIOS RTM + Level 1 Directory header signed with BIOS private key" to level 1 Entry 7
3. Fill the "BIOS RTM + Level 1 Directory header + Level 2 Directory header signed by BIOS private key" to level 2 Entry 7

4.5.5 SBIOS Implementation Details

SBIOS need implements below steps in order to support "PSP initiated crisis recovery"

1. Implement 2 level PSP & BIOS directory XML, reference "AgesaModulePkg\AMDTTools\NewPspKit\Sample" of PI release package for details
2. Protect "Fixed region" during BIOS update
3. Protect "Embedded Firmware Structure" during BIOS update
4. Allow "PSP updateable region" be updatable during recovery path
5. Reserve appropriate size for "Updatable region" for future firmware size growth
6. AGESA PSP module will install PSP recovery required PPI (gAmdPspRecoveryDetectPpiGuid), SBIOS code need notify the PPI, and proceed on BIOS recovery path.

4.5.6 A/B Recovery Changes for Family 19h Models 40h-4fh Onward

The Boot ROM on Family 19h Models 40h-4Fh supports A/B recovery, resulting in changes to the PSP ROM image layout, mainly to put the previously read-only PSP L1 directory contents under A/B recovery.

With the new layout, all binary contents, such as PSP stage 1 bootloader, AMD root key, etc. are now moved into each partition and can be recovered if one copy is corrupted. The PSP L1 directory contains no firmware contents, only pointers to partitions.

The PSP L1 directory itself is not part of A or B partition, but it has an identical backup copy so the system can recover from a corruption of the L1 directory.

4.5.6.1 Structure Changes

EFS Change

- Existing EFS entry (0x14) continues to point to the PSP L1 directory, using an SPI ROM offset.
- New EFS entry (0x2C) points to the backup copy of the PSP L1 directory, using an SPI ROM offset.

PSP L1 Directory Change:

- A PSP L1 directory can have at most 32 entries.
- It contains entries (type 0x48 for partition A, and type 0x4A for partition B) for all SoCs, with each entry pointing to an ISH using a SPI ROM offset.

Image Slot Header (ISH) Addition

- AMD Family 19h Models 40h-4Fh has an added structure, ISH. The PSP L1 directory entry points to ISH, and ISH points to the partition PSP L2 directory. Table 27 shows the ISH structure definition.
- The ISH contains metadata associated with a partition, but **not deemed as part of a partition**.
- Each ISH must take a separate flash block, so that it can be changed separately by the flashing tool or the BIOS update function.

Table 27. ISH Structure Definition

Offset (Hex)	Size (Bytes)	Description/Purpose
0x00	4	Checksum, 32-bit CRC value of the items below this field. Fletcher's checksum algorithm is used for CRC calculation.
0x04	4	Boot priority, value 0xFFFF_FFFF for partition A, 1 for partition B, value 0x0 means unbootable.
0x08	4	Update Retry count, BIOS image always set this as 1.
0x0C	1	Glitch Retry count, BIOS image always set this as 0.
0x0D	3	Reserved
0x10	4	Location, ROM offset of PSP L2 directory table for current slot image.
0x14	4	PSP ID of the SoC that this image slot is for. Refer to Table 12, "Valid PSP IDs per Program" on page 53.
0x18	4	SlotMaxSize, Maximum image size allowed to program into the slot.
0x1C	4	Reserved

Specifically, the Location field inside an ISH continues to use a SPI ROM offset. This prevents it from being treated as part of the pointed partition.

A BIOS build should verify that the 0x48 entry ISH has a higher boot priority than the 0x4A entry for a given SoC.

It is recommended that the BIOS build hardcode 0x48 boot priority to 0xFFFF_FFFF, and 0x4A boot priority to 2, all boot retry count to 1, and glitch retry count to 0.

BIOS Image Layout

The following must be in separate SPI ROM blocks (so that they can be changed separately):

- EFS
- PSP L1 directory
- PSP L1 directory backup copy
- Each partition's ISH
- Each partition (PSP L2 + BIOS L2 directories and contents)

4.5.6.2 Boot Flow Change

The change to the boot flow is highlighted in the following sequence:

1. Bootcode finds EFS as before
2. Bootcode locates the 1st PSP L1 entry (A)
3. Bootcode retrieves the pointed PSP L1 directory
4. Bootcode checks sanity & checksum of the PSP L1 directory. If it fails:
 - a. If already using the last entry in EFS, stall.
 - b. Otherwise, record the error, switch to the next entry in EFS, and restart from step 3.-----bootcode has found a good PSP L1 directory-----
5. Bootcode goes through each PSP L1 directory entry and loads the corresponding ISH.
6. Bootcode compares newly loaded ISH PSP ID with the current SoC PSP ID.
 - a. If multiple matches exist, the bootcode selects the one with the highest boot priority and not marked as unbootable in the warm reset persistent unbootable mask (UBM) register.
 - b. If no match, stall.
7. Bootcode updates the current slot record, and continues to boot (no change).

4.5.6.3 Recovery Process Change

The recovery update normally only updates a specific partition. This means the following structure/sections are not touched (treated as read only) during recovery update:

- EFS
- PSP L1 directory and backup copy

The update utility re-programs everything inside a partition (PSP L2 directory, BIOS L2 directory, and their contents) as before.

The metadata included in a partition's associated ISH are mostly static values on AMD Family 19h Models 40h-4Fh, except the Location field, which does not change when copying the contents from partition B to A. This means ISH does not need to change during a recovery update process.

However, an ISH might have been corrupted and triggered the recovery. This means the recovery update process must check the repaired partition's ISH contents and potentially re-program it if corrupted.

4.5.6.4 Corruption Reporting

PSP firmware or BIOS reports corruption events into BIOS RAM 78h-7Fh. The log format is slightly changed to cover the extra corruption source, as shown in Table 28.

Table 28. Corruption Reporting Log Format

Offset	Sub Offset	Description
[15:0]	[7:0] entry_type [11:8] instance [15:12] subprogram	Entry subprogram, instance and type ID of the corrupted binary that caused recovery.
[19:16]	PSP directory level	b'0000 - not used b'0001-- The entry is from PSP directory L1 b'0010-- The entry is from PSP directory L2 b'0011--The entry is from BIOS directory L2 b'0100 - PSP L1 directory header b'0111 - Image Slot Header
[21:20]	Reserved	Should be 0
[24:22]	Partition number	Which partition this log is from
[31:25]	Reserved	Should be 0

4.6 PSP/BIOS Directory Upgrade Progress

After BIOS completes the post, the PSP will be in the idle state, do not rely on any FW image that exists in the PSP/BIOS directory, except PSP Non-volatile storage. So PSP to X86 SMI interface needs be disabled, and X86 needs to acquire the SMI register mutex prior to flash upgrade.

4.7 Firmware Anti-rollback BIOS Requirement

PSP platform firmware anti-rollback (FAR) feature is designed to prevent rollback of platform level firmware to older version that are deemed vulnerable from a security point of view. It effectively provides a revocation mechanism for firmware operating in field. Basic requirements are that this feature, at a minimum, should apply to critical firmware carried in the SPI flash of a system.

FAR utilizes a table which contains the required security patch level (SPL) of relevant firmware entries that are desired to be protected from rollback. The SPL is independent of the firmware version itself, and is only meant to be updated when vulnerabilities are found and subsequently fixed.

The table itself will also carry a SPL value, and this SPL value will be enforced against a fuse in the SOC, chaining the root of trust into the hardware itself. This prevents rollback of the table. Updates to the table itself, will effectively revoke older versions of the table and hence older versions of the firmware listed in the table itself. The number of tables will be limited by the number of fuses allocated, and can differ from program to program.

Upon boot, each component that loads and validates firmware is responsible for enforcing the SPL of the loaded firmware. For example, if Boot ROM loads and validates an off-chip bootloader, then the Boot ROM is expected to enforce the SPL of the off-chip bootloader against the corresponding entry listed in the table.

Platform BIOS requirements:

1. FAR requires a BIOS design where a backup or recovery partition is applicable, the backup partition must be kept in sync with the main partition. For client products, platform BIOS must enable A/B recovery to enable FAR.
2. Set `gEfiAmdAgesaModulePkgTokenSpaceGuid.PcdAmdPspAntiRollbackLateSplFuse` in `AgesaModulePspPkg.dec` to TRUE to enable the FAR feature in BIOS
3. Keep `gEfiAmdAgesaModulePkgTokenSpaceGuid.PcdAmdPspAntiRollbackInitialSpl` in `AgesaModulePspPkg.dec` as default value 0 if you do not need to specify initial SPL value. Otherwise, set it to the value as you want.

When initial SPL is set to 0, the BIOS asks PSP to check whether the SPL fuse should be upgraded. If the SPL value in the SPL table is greater than SPL fuse in the CPU, the SPL fuse is be upgraded.

When initial SPL is set to a non-zero value, the initial SPL is fused to CPU at the first boot. In subsequent boots, when FAR has been enforced, the BIOS does not ask PSP to upgrade SPL fuse, so SPL fuse is kept unchanged.

PSP firmware only honors the BIOS-specified value (initial SPL value) when the following requirements are met:

- FAR enablement fuse is 0 (FAR not enabled)
- SPL value provided by BIOS meets the criteria:
(`SPL_FUSE` <= `SPL_BIOS_INPUT` <= `SPL_PRESENT_VALUE`)
- `SPL_FUSE`: SPL value read from fuse (0 by default if FAR is not yet enabled)
- `SPL_BIOS_INPUT`: Requested value to be fused (initial SPL value)
- `SPL_PRESENT_VALUE`: SPL value detected by PSP firmware in current operating stack
- `SPL_BIOS_INPUT` to be provided with existing BIOS enablement command `BIOS_CMD_LATE_SPL_FUSE` (0x2D)

- If SPL_BIOS_INPUT is 0, then PSP firmware uses SPL_PRESENT_VALUE instead. This command must be sent prior to BIOS_CMD_BOOT_DONE.

When the FAR feature is enabled, sending the fuse command updates the fuse to match the SPL_PRESENT_VALUE. The SPL_BIOS_INPUT value is ignored.

4. Include the security patch level (SPL) table in BIOS as PSP directory entry.

- include the SPL table as PSP directory entry 0x55 for AMD Family 17h Models 60h-67h, AMD Family 19h Models 50h-5Fh, and AMD Family 19h Models 10h - 1Fh and onwards.
- For programs after AMD CPU Family 17h Model 60h-67h and AMD CPU Family 19h Models 50h-5Fh, include two SPL tables, PSP directory entry 0x55 used by Boot ROM, and PSP directory entry 0x56 used by off-chip bootloader. For sever programs that support FAR, only entry 0x55 is used.

To find out if FAR is enabled, send the command MboxBiosCmdHSTIQuery (0x14) to PSP. If BIT 7 is set, FAR is enabled in the system.

4.8 How to Add RPMC Support in BIOS

- Add PSP entry 0x54 for PSP NVRAM
Entry point to PSP NVRAM for RPMC
- Implement two (2) PSP to BIOS commands:
 - a. Increment Monotonic Counter
This command is used by the SPI Controller to increment the Monotonic counter by 1 inside the SPI Flash.
 - Command id: 0x88
 - Function call: P2CmboxSpiRpmcIncMc
 - b. Request Monotonic Counter
This command is used by the SPI Controller to request the Monotonic counter value inside the SPI Flash.
 - Command id: 0x89
 - Function call: P2CmboxSpiRpmcReqMc
- When fTPM and RPMC is enabled, remind user to clear PSP NVRAM if entry 0x54 is corrupted.

4.9 Use AmdPspPsbFusingLib to Customize PSB Fusing

The PCD PcdAmdPspAutoPsb has been provided to set the PSB related fuse automatically. This is the suggested and default method to enable the PSB.

A library AmdPspPsbFusingLib can be used to manually control when to enable PSB. This library provide two interfaces:

- IsPsbFusingRequired: Check if PSB fusing is required.
- DoPsbFusing: Set PSB fusing.

The following diagram shows the flow.

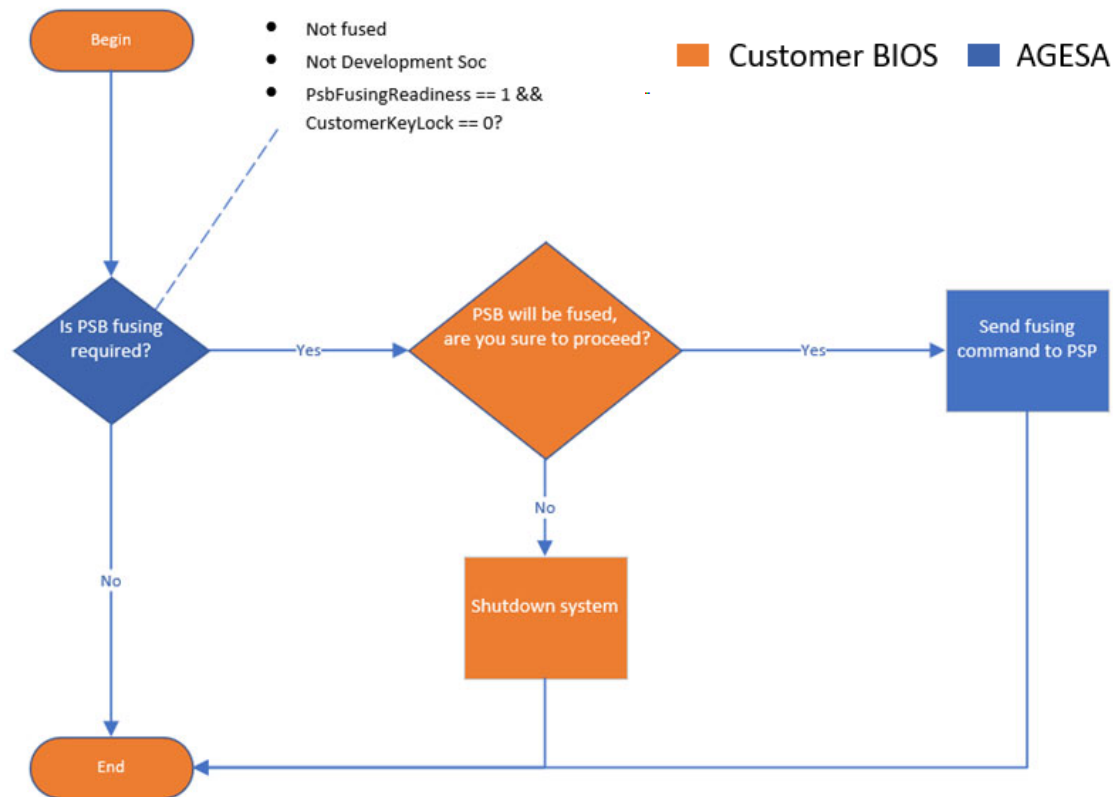


Figure 7. PSB Fusing Process Flow

Note that AGESA only provides library interfaces. Customers who want to manually control the PSB fusing need to integrate this library with their own business logic.

Chapter 5 X86 BIOS S3-Resume Path Handling

5.1 BIOS S3 Transition Flow on ACPI Aware OS

BIOS implements the following support for S3 transitions:

- a. During the cold boot path, the BIOS sets up an SMM trap on the Sleep transition.
- b. When the OS transitions to the S3 path, it writes the SLP_TYP to the PmControl IO register to transition the platform to the S3 state.
- c. This action generates an SMM trap and the BIOS SMM handler gets control.
- d. The BIOS SMM routine sends mailbox command "MBoxBiosCmdSxInfo" to the PSP firmware and this mailbox command informs PSP about the final Sx sleep state the system will transition to.
- e. BIOS then waits for the PSP to finish the pre-S3 transition items.
- f. After the PSP acknowledges mailbox command completion, the BIOS finally writes the SLP_TYPE value to the PmControl IO register to transition the system to the sleep state.

5.2 BIOS S3 Resume

Under secure mode the PSP firmware restores the memory during resume. Hence, the DRAM is already available when the x86 cores come out of reset on resume.

The availability of DRAM provides an opportunity to optimize the BIOS resume path. For example, there is no need to perform Cache as RAM initialization. Also, a new CPU MSR is added and any write to this MSR results in CPU context (MSR, microcode, etc.) being automatically saved in protected fenced memory. During resume from sleep transition, the CPU core automatically restores to this CPU context saved in fenced memory; and x86 cores immediately execute from the resume vector that was passed as part of write to MSR during cold boot. The availability of DRAM and properly restored CPU context at x86 reset time provide an opportunity to improve BIOS resume time.

The existing UEFI BIOS currently does not comprehend this kind of S3 resume. Conventionally, BIOS PEI driver is expected to run from ROM code and perform memory restore. Historically, the reset addresses for x86 cold boot and S3-resume were the same (i.e., 0xFFFFFFFF0). Hence, during the resume path the BIOS code executed the same PEI driver in SPI space that it would execute during cold boot. In this case the BIOS resume vector can be the DRAM location and the resume path can be optimized to make use of this fact.

5.2.1 Modified Conventional Resume

In this path, the x86 resume from sleep path will begin the x86 core executing code from DRAM and then jump to ROM code to continue conventional resume. During the cold boot the S3 reset vector

code is copied, via write to new MSR, at OS visible memory or protected SMM space. In one such implementation the BIOS SMM function perform special MSR write to save CPU context and resume vector in SMM space as explained before. Just like above case the BIOS SMM code will write to the MSR so that on resume from S3 when PSP firmware restores DRAM and x86 microcode restores CPU context from memory the x86 core will jump to SMM code on X86 reset. This SMM function will update the SMM save area of each core and resume out of SMM to jump to alternate BIOS ROM entry point in PEI Core code. The SMM save state that will be updated by SMM resume driver includes information regarding location of PEI GDT table, RIP and RSP for the code outside SMM (e.g., the GDT location in SMM save area is patched to point to GDT table in ROM) in addition stack pointer (RSP) location in SMM save area will be set to use BIOS reserved memory locations as stack and finally the RIP in SMM state is patched to jump to alternate PEI entry point in PEI volume. The SMM code may further authenticate PEI volume, if OEM desires; after the basic setup the SMM code will resume (RSM) out of SMM mode and the control will be transferred to alternate entry point of PEI core. The alternate PEI code will then use DRAM and not cache (as RAM) for stack and continue with regular resume path and later handoff control to OS. Other enhancements can be added in the PEI kernel code on resume path.

For details, please refer to Appendix C, “Modified Conventional Resume S3 (SMM->SEC->PEI) Design Guideline” on page 139.

Chapter 6 TPM Software Interface

The PSP software solution-stack offers firmware based TPM 2.0 services based on Microsoft whitepaper — “Trusted execution environment ACPI profile.”

The TPM software interface is being defined by the TCG’s PC client work group and will be ratified as part of TCG PC Client Specific Platform TPM Specification; the current draft TPM software interface section of this specification is referred to as the Command/Response Buffer Interface and is defined in the TPM Command/Response Buffer Interface w/Locality Support, Version 0.56, DRAFT, 01-09-2013. This specification is undergoing review and is expected to be ratified as a TCG standard defining the TPM 2.0 interface.

6.1 TPM 2.0 Command/Response Buffer Interface

This interface defines a control area structure as shown in Table 29. The physical address of this control area is specified in a TPM 2 ACPI table set up by the BIOS.

The command/response buffers used in this interface are allocated in DRAM by BIOS after the memory is available.

Table 29. Control Area Layout

Field	Byte Length	Offset	Description
Miscellaneous	4	00h	Used for Power Transition
Status	4	04h	SET by the TPM to indicate an error condition or that it is in idle state
Cancel	4	08h	SET by software to abort command processing
Start	4	0Ch	SET by software to indicate that a command is available for processing.
Interrupt Control	8	10h	Reserved. (Must be zero.)
Command Size	4	18h	Size of the Command buffer
Command Address	8	1Ch	This field contains the physical address of the command buffer.
Response Size	4	24h	Size of the Response Buffer
Response Address	8	28h	This field contains the physical address of the response buffer.

This interface is described under section 1.2 of TPM Command/Response Buffer Interface w/Locality Support, Version 0.56, DRAFT, 01-09-2013 in greater details including how these fields are controlled in submitting commands and receiving responses, canceling the commands with

comprehensive state and sequence diagrams. As such, those details are not repeated here and specification should be consulted for implementation.

6.2 AMD Implementation of TPM 2.0 Interface

TPM interface definition structure as defined in the TPM Command/Response Buffer Interface w/ Locality Support, Version 0.56, DRAFT, 01-09-2013 that includes the interface identifier, control area and control area extension are mapped to a set of CPU-PSP MMIO message registers, as shown in Figure 8.

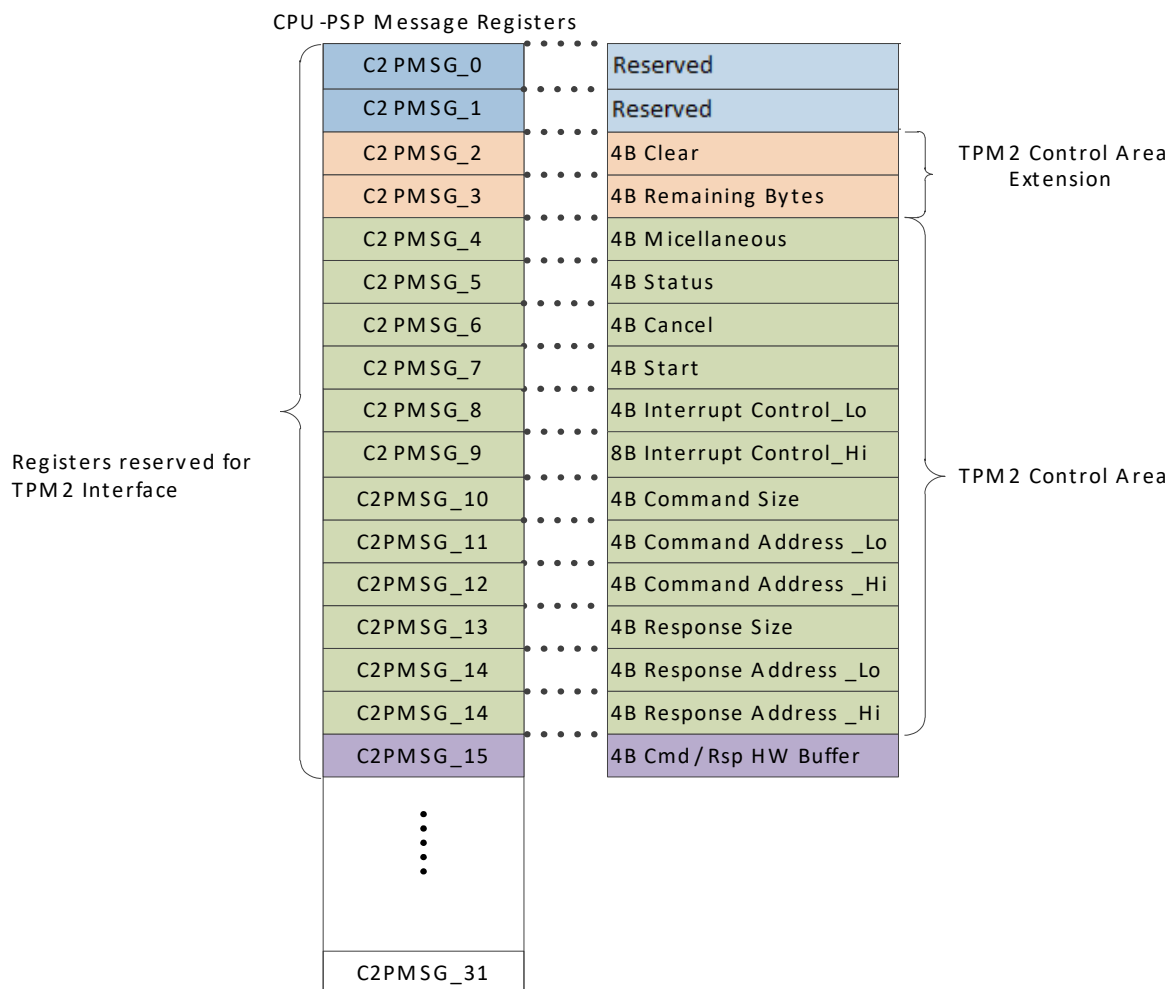


Figure 8. TPM2 Command/Response Buffer Interface

The TPM interface identification structure as defined in the TPM Command/Response Buffer Interface w/Locality Support, Version 0.56, DRAFT, 01-09-2013, is common to both TPM 1.2 and TPM 2.0 is expected to be located at a well-known fixed physical address. However, in AMD Family

16h Models 30h–3Fh processor implementation of TPM 2.0, it is not possible for this structure to be placed at a fixed address as these fields are mapped to CPU-PSP mailbox registers as shown in Figure 8.

Therefore, the BIOS fTPM driver that initializes and uses this interface is required to be aware of the AMD specific implementation on where this structure is placed.

After BIOS has completed DRAM training it initializes the Command/Response Buffer interface by:

- Building the TPM2 static ACPI table, and set the start method to 2. (Uses the ACPI Start method.)
- Reserving TPM 2.0 command/response buffers in DRAM,
- Updating the ACPI control area with the physical address of the command/response buffers.
- Implement TPM ACPI Start Method reside in the _DSM.

Note: *TPM 2.0 interface and TPM 2.0 commands are only supported after the DRAM becomes available.*

6.2.1 Disable fTPM

Below describes the steps of disabling fTPM support by AGESA package.

If required to disabled permanently:

1. Remove the "AmdPspFtpmPei.inf" & "AmdPspFtpmDxe.inf" from AGESA .inc.fdf file. It will force AGESA not declare any HW resource for fTPM
2. Not report the fTPM ACPI Object to OS, including both ACPI table and ASL method related.

If required to disabled during post time:

1. Set PcdAmdPspFtpmEnable PCD token to FALSE, PSP fTPM driver will exit immediately after checking the PCD value, no HW resource will be declared.
2. Not report the fTPM ACPI Object to OS, including both ACPI table and ASL method related.

6.2.2 Clear fTPM NVRAM

fTPM implementation relies on device unique key (Different per processor) to encrypt TPM context records that are stored in the NVRAM. Those records are generated the first time fTPM was enabled and it stays in the NVRAM for the entire life of the system. However, there are several cases required to clear fTPM NVRAM: NV region corruption, NV structure updated, and processor swap. When the user decides to upgrade/swap their CPU with fTPM enabled for any purpose, fTPM is not able to decrypt the record as the device unique key is different. There is no way for TPM protocol to clear those context records as TPM_Clear can't clear any TPM context. In this case, the only way to allow the user to continue to use TPM with the new CPU is to

clear TPM non-volatile region. This allows NVRAM to be empty and allows TPM to re-provision all the context records for the new CPU.

More specifically, PSP firmware will return with

PSP_CAP_TPM_REQ_FACTORY_RESET bit set for C2P mailbox

MboxBiosCmdPspQuery command. For additional information, see

“MboxBiosCmdPspQuery (MboxCmd = 0x05)” on page 99. After AGESA PSP

module detected this bit set, it will publish *gAmdPspFtpmFactoryResetPpiGuid* PPI.

Platform BIOS can take the further step of handling this PPI, for example, it can pop up a warning message and allow user to clear TPM NV region (PSP Directory Entry 0x04). The Warning message may be shown as below:

New CPU installed, fTPM/PSP NV corrupted or fTPM/PSP NV structure changed.

Press **Y** to reset fTPM, if you have BitLocker or encryption enabled, the system will not boot without a recovery key

Press **N** to keep previous fTPM record and continue system boot, fTPM will NOT enable in new CPU, you can swap back to the old CPU to recover TPM related Keys and data.

PSP Directory Entry type 0x54 is needed for multiple uses: RPMC and TEE persistent objects, etc, are closely tied with fTPM NV RAM. During a post, the AGESA PSP module will send the *MboxBiosCmdPspCapsQuery* command to the PSP FW to query the health of this region. If corruption is detected, the AGESA PSP will set PCD *PcdAmdPspNvramClearRequest* to TRUE. After the Platform BIOS detects that this PCD has been set, it must clear both PSP Directory Entry type 0x4 fTPM NV RAM and 0x54 PSP NV RAM because this operation clears the TPM NV data that has user confidential data stored. The warning message above is still needed to highlight to the impact.

Chapter 7 BIOS PSP Mailbox Interaction

The BIOS-to-PSP and PSP-to-BIOS communication interfaces are implemented as mailboxes mapped to a set of MMIO CPU-PSP registers and to a portion of SMM memory area reserved for PSP by BIOS respectively, as shown in Figure 9:

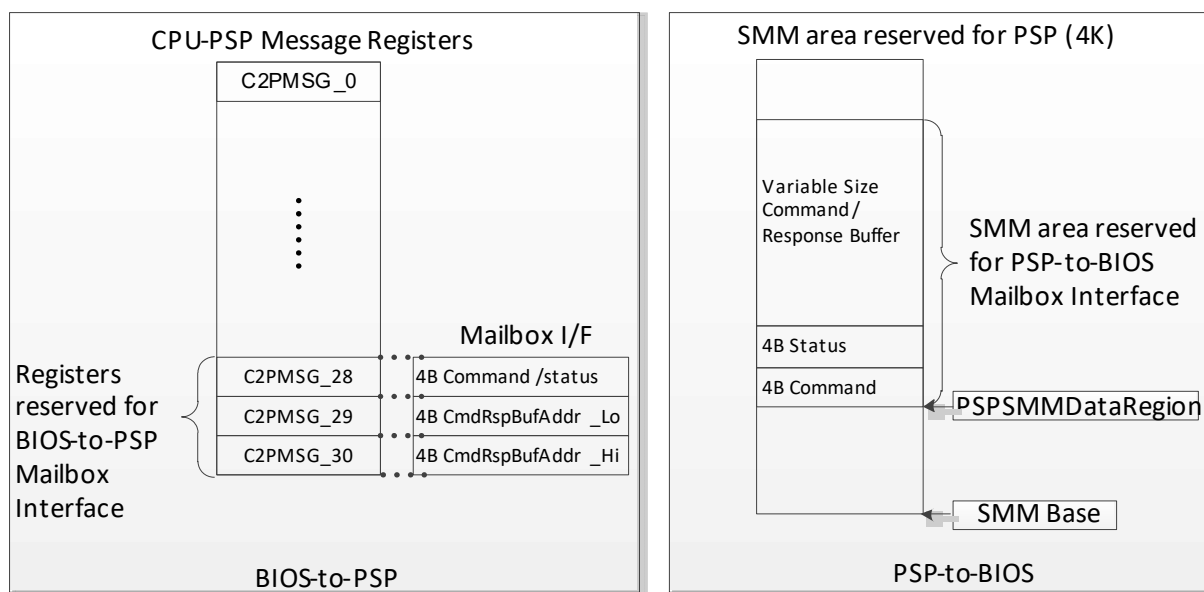


Figure 9. BIOS-PSP Mailbox Interface

7.1 BIOS to PSP Mailbox

The mailbox interface consists of 4-byte command/status and address fields; and a command/response buffer.

The CmdRspBufAddr_Lo and CmdRspBufAddr_Hi fields are set to the bits 0:31 and bits 32:63 of a 64-bit physical address of the command/response buffer in DRAM.

The command/response register is a 32-bit value initialized to zero by the target. It consists of Command ID (bits [23:16]), Status (bits [15:0]) fields and Ready flag (bit #31). The host submitting the command, first writes the command data into the command buffer and then writes the 32-bit command identifier to this register to indicate a new command has been placed in the command buffer (the Ready flag is cleared). It then waits for the Ready flag to be set by the target indicating the command completion.

The target processing the command sets Ready flag when it has completed processing the command.

The host, after writing to the command field, waits for the target to set Ready flag and when it is set reads the status bits and response data from the command/response buffer.

Figure 10 shows the command execution sequence over this mailbox interface.

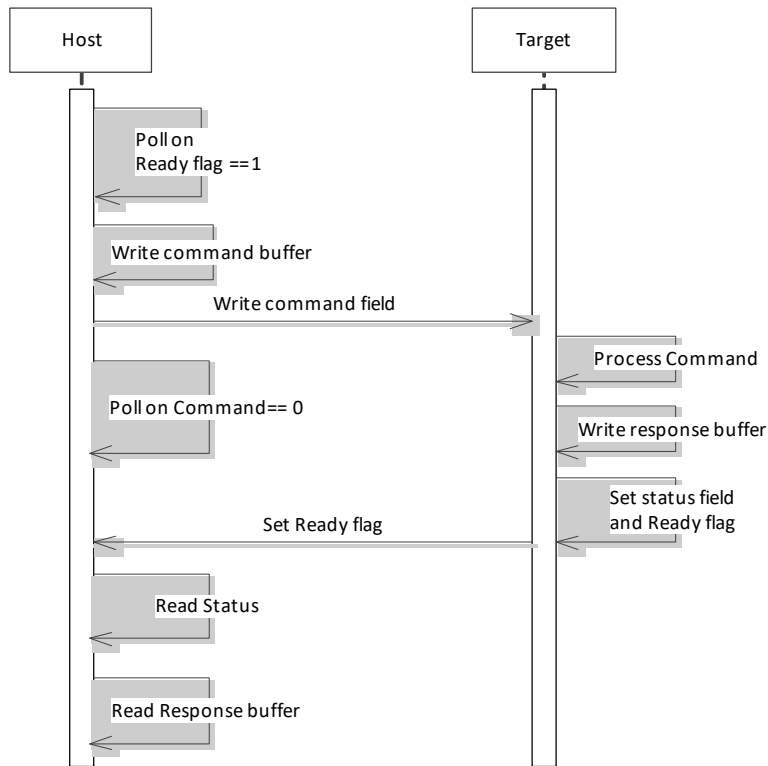


Figure 10. BIOS-PSP Mailbox Command Execution Sequence

PSP FW, in its role as a target on the BIOS-to-PSP communication interface, can enable interrupts to be generated on BIOS writes to the command register.

Table 30 self-reference. defines the bit-fields of the status field:

Table 30. BIOS-PSP Mailbox Status Register Bit Fields

Field Name	Bit Index	Description/Purpose
Ready	31	Set by the target to indicate the mailbox interface state. 0 — Not ready to handle commands (or handling previous command) 1 — ready to handle next command
Recovery	30	Set by the target to indicate that the host has to execute FW recovery sequence

Table 30. BIOS-PSP Mailbox Status Register Bit Fields (Continued)

Field Name	Bit Index	Description/Purpose
ResetRequired	29	Set by the target to indicate that the host must execute warm reset if FW corruption is detected.
Command ID	[23:16]	Command ID set by host
Status	[15:0]	Set by the target to indicate the execution status of last command

Table 31 defines the set of BIOS-to-PSP commands currently defined:

Table 31. BIOS-to-PSP Mailbox Commands

Command	Value	Description/Purpose
MboxBiosCmdSmmInfo	0x02	Provides details on SMM memory area reserved for PSP. It includes the physical addresses of SMM Base and PSP SMM data region and the length of PSP SMM data region.
MboxBiosCmdSxInfo	0x03	Notification that the platform is entering S3-suspend state.
MboxBiosCmdRsmInfo	0x04	Information on BIOS Resume Module stored in SMM memory includes the BIOS resume vector and size of the resume code.
MboxBiosCmdPspQuery	0x05	Command to get the list of capabilities supported by PSP FW. This is used to communicate if fTPM is supported in PSP FW.
MboxBiosCmdBootDone	0x06	Notification that BIOS has completed BIOS POST.
MboxBiosCmdHSTIQuery	0x14	Command to get HSTI bit field representing the security state of the SOC from the PSP.
MboxBiosCmdGetVersion	0x19	Get FW version.
MboxBiosCmdSetFuse	0x1A	BIOS sends this command to set Field Programmable fuse; only enabled in special released FW.
MboxBiosCmdLockDFReg	0x1B	BIOS will send this command to lock DF registers.
MboxBiosCmdClrSmmLock	0x1C	Command to clear SMMLock register in C6 private memory region.
MboxBiosCmdSetApCsBase	0x1D	BIOS will send the CS BASE value for AP threads.
MboxBiosCmdKvmInfo	0x1E	KVM required information.
MboxBiosCmdLockSpi	0x1F	BIOS will send this command to lock SPI; X86 must be in SMM mode when sending this command.
MboxBiosCmdScreenOnGpio	0x20	Report the FCH GPIOs for early turn on eDP panel in S0i3.
MboxBiosCmdSpiOpWhiteList	0x21	BIOS sends SPI operation whitelist to lock SPI; X86 must be in SMM mode when sending this command; only used in server product
MboxBiosCmdPsbAutoFusing	0x21	PSP will set the PSB related in the field; used only in non-server products.
MboxBiosCmdRasEinj	0x22	BIOS sends RAS Error Injection action.
MboxBiosCmdStartArs	0x24	Command to Start ARS for RAS feature.

Table 31. BIOS-to-PSP Mailbox Commands (Continued)

Command	Value	Description/Purpose
MboxBiosCmdStopArs	0x25	Command to Stop ARS for RAS feature.
MboxBiosCmdSetBootPartitionId	0x26	BIOS sends this command to PSP to write the ACTIVE_BOOT_PARTITION_ID register.
MboxBiosCmdPspCapsQuery	0x27	BIOS checks PSP NVRAM health.
MboxBiosCmdDrtmInfoId	0x41	BIOS sends this command to PSP about the information DRTM feature required.
MboxBiosCmdLaterSplFuse	0x2D	BIOS sends this command to PSP for SPL fuse for anti-rollback feature.
MboxBiosCmdDtpmInfo	0x2E	BIOS-to-PSP: Command to get dTPM status and event log.
BIOS_CMD_VALIDATE_MAN_OS_SIGNATURE	0x2F	BIOS-to-PSP: Validate signature of manageability OS image based on header passed by BIOS.
MboxBiosCmdLockFCHReg	0x30	BIOS-to-PSP: BIOS sends this command to lock FCH PM and IOMux registers.
BIOS_CMD_GET_DRTM_INFO	0x31	BIOS-to-PSP: Queries updated dRTM information in post dRTM phase.
MboxBiosCmdSetRpmcAddress	0x39	BIOS-to-PSP: Bios sends this command to PSP to decide which RPMC address to use. Only used in product line. A warm reset should be issued after receiving BIOS_MBOX_OK (0) from PSP tOS, otherwise BIOS does nothing.
MboxBiosCmdLockGPIO	0x3A	BIOS-to-PSP: BIOS send this command to PSP to lock GPIO.
MboxBiosCmdSendIvrsAcpiTable	0x3F	BIOS-to-PSP: BIOS sends IVRS buffer to PSP, PSP saves it, then AMDSL uses another command to retrieve it back to the buffer.
MboxBiosCmdTa	0x40	Send command to TA.
BIOS_CMD_ACPI_RAS_EINJ	0x41	BIOS-to-PSP: Enables/disables ACPI-based RAS EINJ feature.
MboxBiosCmdQueryTCGLog	0x42	BIOS-to-PSP: Queries TCG Log.
MboxBiosCmdQuerySplFuse	0x47	BIOS-to-PSP: Gets the current value of the SPL_F (FW_ROLLBACK_CNT) fuse value.

7.1.1 BIOS to PSP Mailbox Commands

To send mailbox command to PSP, the BIOS builds the command/response buffer in system memory and updates the 64 bit MMIO CmdRspBufAddr pointer to point to this command/response buffer in the memory. BIOS then writes the mailbox command, as listed below, in the MMIO command register and waits for the command register to return to zero (as controlled by PSP firmware). After the command is processed, BIOS reads the status of the mailbox operation from the mailbox status register as well as the status value in the command/response buffer.

The generic format of the command/response buffer is below

```
typedef struct {
```



```

    UINT32    TotalSize;
    UINT32    Status;
} MBOX_BUFFER_HEADER;

typedef struct {
    MBOX_BUFFER_HEADER    Header;
    COMMAND_SPECIFIC_BUFFER    Buffer;
} MBOX_COMMAND_RESPONSE_BUFFER;

```

Each command/response buffer starts with a standard mailbox buffer header. The field TotalSize is set to the size of command/response buffer. The field Status is updated by PSP firmware to reflect the status of the command that is being handled by PSP firmware. The standard mailbox buffer header is followed by a command-specific buffer that is different per each mailbox command.

7.1.1.1 MboxBiosCmdSmmInfo (MboxCmd = 0x02)

After the SMM environment is ready the PSP SMM driver sends this mailbox command to PSP firmware. The PSP firmware will use information supplied via this command to later trigger SMM, and request BIOS services for the PSP NVRAM region in SPI space. The command/response buffer is of structure type MBOX_SMM_BUFFER as defined below:

```

/// Define structure of SMM_TRIGGER_INFO
typedef struct {
    UINT64    Address;                ///< Memory or IO address (Memory will
be qword, IO will be word)
    UINT32    AddressType;            ///< SMM trigger typr - Perform write
to IO/Memory
    UINT32    ValueWidth;             ///< Width of value to write (byte
write, word write,...)
    UINT32    ValueAndMask;           ///< AND mask of value after reading
from the address
    UINT32    ValueOrMask;            ///< OR Mask of value to write to this
address.
} SMM_TRIGGER_INFO;

typedef struct {
    UINT64    Address;                ///< Memory or IO address (Memory will
be qword, IO will be word)
    UINT32    AddressType;            ///< SMM trigger typr - Perform write
to IO/Memory
    UINT32    ValueWidth;             ///< Width of value to write (byte
write, word write,...)
    UINT32    RegBitMask;             ///< AND mask of value after reading
from the address
    UINT32    ExpectValue;            ///< OR Mask of value to write to this
address.
} SMM_REGISTER;

typedef struct {

```

```

    SMM_REGISTER SmiEnb;                ///< Register information for SmiEnb
    SMM_REGISTER Eos;                   ///< Register information for EOS
    SMM_REGISTER FakeSmiEn;             ///< Register information for FakeSmiEn
    SMM_REGISTER Reserved[5];
} SMM_REGISTER_INFO;
///
/// structure of ReqBuffer for MboxBiosCmdSmmInfo mailbox command
///
typedef struct {
    UINT64          PspSmmDataRegion;    ///< PSP region base in Smm space
    UINT64          PspSmmDataLength;    ///< Psp region length in smm space
    SMM_TRIGGER_INFO SmmTrigInfo;        ///< Information to generate SMM
    SMM_REGISTER_INFO SmmRegInfo;        ///< Information describe the SMM
register information
} SMM_REQ_BUFFER;

```

The command/response buffer structure starts with a standard mailbox header.

SmmBase—This field provides the location of SMM base.

SmmLength—This field provides the length of SMM space.

PspSmmDataRegion—The PSP SMM driver allocates SMM space for PSP use and this field provides the location of SMM space that is carved out for PSP use.

PspSmmDataLength—This field provides the length of the PSP region in SMM space. PSP firmware must not access SMM space beyond this allotted space.

SMM_TRIGGER_INFO—Provides information on how the PSP firmware can trigger an SMI. There are various ways to trigger SMI: write to MMIO, I/O, or PCI config space. This structure provides the following information:

Address—The Physical address where PSP needs to write to trigger SMI

AddressType—The type to access to trigger SMI; IO (0), MMIO (1) or PCI (2)

ValueWidth—Write length: Byte (0) , Word (1) , Dword (2), Qword (3) on Address

ValueAndMask—AndMask

ValueOrMask—OrMask

SMM_REGISTER_INFO—describes the SMM register information which PSP FW needs to check or polling before issues a SMI.

To trigger an SMI, the PSP firmware first reads from the address location as defined by AddressType and performs an “AND” operation based on ValueAndMask followed by an “OR” operation based on ValueOrMask and writes it back to the Address location.

7.1.1.2 MboxBiosCmdSxInfo (MboxCmd = 0x03)

The BIOS SMM driver sends this command right before the system transitions to sleep state. PSP firmware performs any last minute save operations just prior to S3 transition when BIOS sends this command to PSP.

The command/response buffer has the following structure. It starts with the standard mailbox buffer header, followed by the SleepType field that informs the PSP of the sleep state the system is about to transition to.

```
typedef struct {
    MBOX_BUFFER_HEADER Header;
    UINT8 SleepType;
} MBOX_SX_BUFFER;
```

7.1.1.3 MboxBiosCmdPspQuery (MboxCmd = 0x05)

BIOS sends this command to PSP to find the capabilities offered by PSP. In response to this command, PSP firmware updates the Capability field in the command/response buffer as shown below:

```
typedef struct {
    UINT32 Capabilities;
} CAPS_REQ_BUFFER;

// Bitmap defining capabilities
#define PSP_CAP_TPM (1 << 0)

typedef struct {
    MBOX_BUFFER_HEADER Header;
    UINT32 Capability;
} MBOX_CAPS_BUFFER
```

The command/response buffer starts with a standard mailbox buffer followed by a Capability field that is updated by PSP firmware. Each bit in the Capability field represents a feature supported by PSP as described in Table 32 on page 99.

Table 32. Bit Definitions for Capability Field

Bit	Name	Description
-----	------	-------------

Table 32. Bit Definitions for Capability Field

0	PSP_CAP_TPM_SUPPORTED	This return code from PSP means fTPM binary and fTPM NVRAM are good. fTPM is fully functional and BIOS/OS can start using them.
1	PSP_CAP_TPM_REQ_FACTORY_RESET	This return code from PSP means that NVRAM region of fTPM got corrupted and BIOS has to clear the whole NVRAM region and reboot the system for fresh start.
2	PSP_CAP_FTPM_NEED_RECOVER	This return code from PSP means that fTPM binary available in SPI-ROM(PSP directory entry) is corrupted and fTPM cannot be loaded, BIOS has to trigger the recovery mode so new image can be flashed to SPI-ROM.

7.1.1.4 MboxBiosCmdBootDone (MboxCmd = 0x06)

BIOS sends this command to PSP before handing off control to the OS. This mailbox command is an indication to PSP firmware to no longer handle any more BIOS mailbox commands other than commands coming from SMM space. After this command is sent, PSP will only handle a mailbox command if the command buffer is within SMM region.

No additional parameters are needed for this command. CmdRspBufAddr points to the mailbox buffer.

7.1.1.5 MboxBiosCmdHSTIQuery (MboxCmd = 0x14)

Command to get HSTI bit field representing the security state of the SoC from the PSP

7.1.1.6 MboxBiosCmdGetVersion (MboxCmd = 0x19), Obsoleted

Command to get FW versions, including PSP firmware, SMU firmware, and PSP AGESA firmware.

7.1.1.7 MboxBiosCmdLockDFReg (MboxCmd = 0x1B)

BIOS sends this command to PSP FW to secure DF related register. This command will be issued when PciEnumerationCompleteProtocol installed.

7.1.1.8 MboxBiosCmdClrSmmLock (MboxCmd = 0x1C)

BIOS sends this command to clear SMMLock Register in C6 private memory region. This command is obsoleted in Family 17h, as unconditionally clear SMMLock on S3 image has been implemented

7.1.1.9 MboxBiosCmdSetApCsBase (MboxCmd = 0x1D)

BIOS sends this command to PSP FW to set AP CS base to a Dram Address, this command currently only used in boot from SPI scenario.

7.1.1.10 MboxBiosCmdKvmInfo (MboxCmd = 0x1E)

BIOS sends this command to KVM engine application running on PSP processor with KVM required information. For more information, please refer to "PID 56053, KVM Integration Guide"

7.1.1.11 MboxBiosCmdLockSpi (MboxCmd = 0x1F)

BIOS sends this command to lock SPI. X86 must be in SMM mode when this command is sent.

7.1.1.12 MboxBiosCmdScreenOnGpio (MboxCmd = 0x20)

In Modern standby enabled system, in order to shorten wakeup time, BIOS send FCH GPIO list for early turn on eDP panel to PSP FW. For more information, see the "PID 56358, Modern Standby BIOS Implementation Guide"

7.1.1.13 MboxBiosCmdSpiOpWhiteList (MboxCmd = 0x21)

In server product, BIOS send command with information about how lock access to the SPI ROM. This information includes:

- "OpCode[7:0]: this is simply the "whitelist" SPI Opcode
- "Options[1:0]

0	0	This OpCode does not require additional check
0	1	This is WRITE command, firmware should make sure the address does not cross cacheline and it is not inside the write protection space
1	0	This is READ command, firmware should make sure the address does not cross cacheline and it is not within the read protection space
1	1	This Opcode is a ERASE command with address. This Option will require a SIZE field. Firmware will check the address in the index FIFO (within SPI controller): <ul style="list-style-type: none"> • Does not cross write protection range • Aligns with SIZE definition accordingly. For example, if the SIZE is 256B, lower 8-bit address must be all 0s.

- "SIZE[N:0]: This field defines the size of ERASE:

000b == 256B,

001b == 512B,

010b == 1KB,

011b == 2KB,

100b == 4KB,

101b == 8KB,

110b == 16KB,

111b == 32KB

And so on If > 32KB is needed

- "Frequency

At this time, only Fixed Frequency is supported. All commands are run at the common slowest speed through index mode

Additionally:

1. Whitelist is static and needs to be part of Platform BIOS.
2. PSP will only accept SPI LOCK if x86 is in SMM mode
3. The earliest the SPI can be locked is after the APCB options are written to SPI ROM.
4. Both a WARM Reset or a Cold RESET will unlock the SPI ROM

7.1.1.14 MboxBiosCmdPsbAutoFusing (MboxCmd = 0x21)

In non-server products, PSP will perform below steps to set PSB related fuse when this command received.

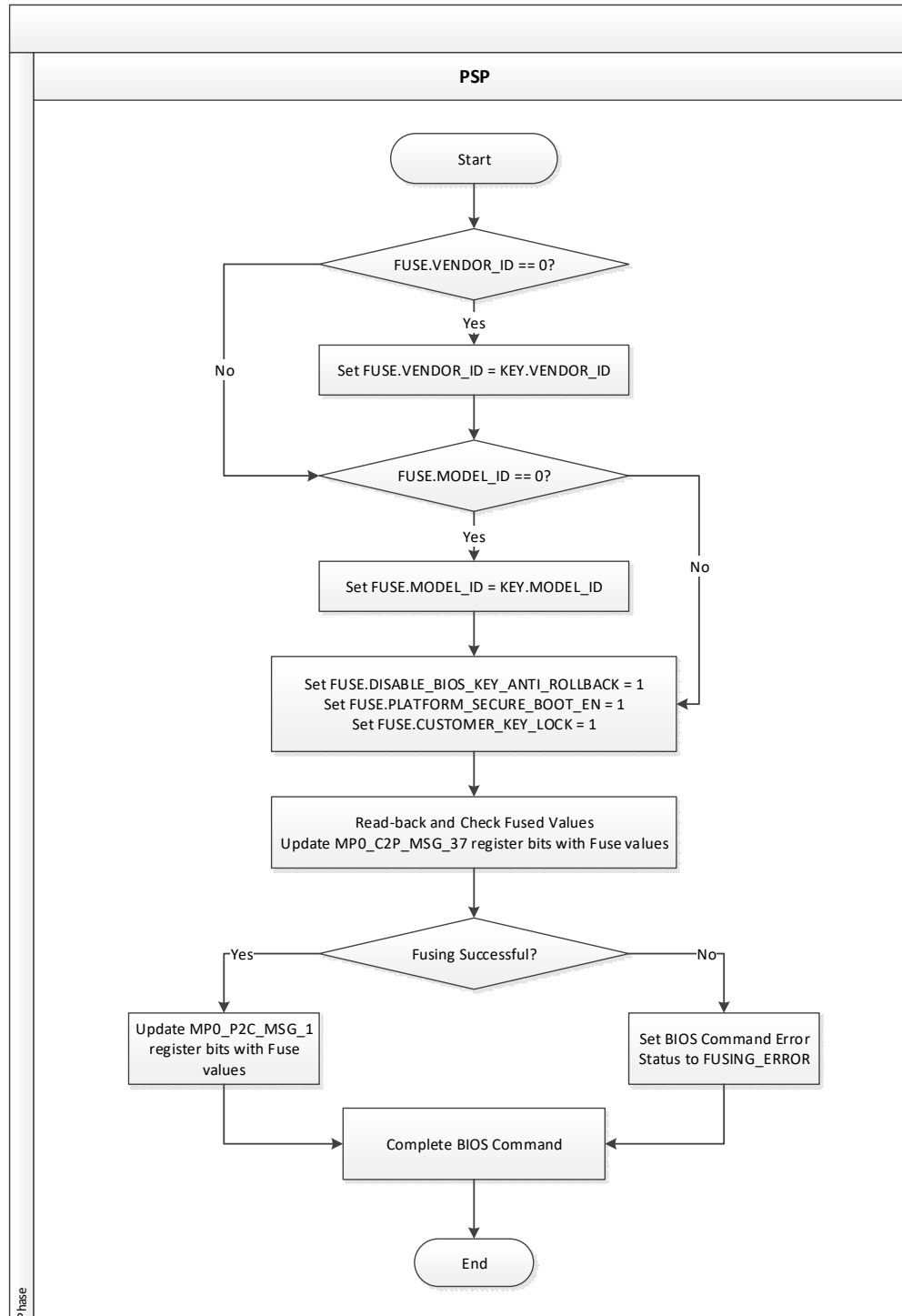


Figure 11. Enable PSB Fusing Command Processing

7.1.1.15 MboxBiosCmdRasEinj (MboxCmd = 0x22)

BIOS sends this command to PSP FW about RAS Error Injection action

7.2 PSP-to-BIOS Mailbox

In the case of PSP-to-BIOS mailbox the command/response buffer starts at offset 0x08 from the PSPSMMDatRegion.

PSP FW, in its role as a host on the PSP-to-BIOS communication interface uses the System Management Interrupt (SMI) mechanism to generate interrupt to BIOS. PSP FW generates the SMI interrupt by writing a pre-defined value either to an IO or Memory address. The value to be written and the type of address are communicated to PSP FW by BIOS through BIOS-to-PSP command MboxBiosCmdSmmInfo. BIOS should also insure StarupAllAPs cannot be interrupted by any asynchronous SMI, including the SMI generated by PSP firmware.

Table 33 defines the set of PSP-to-BIOS commands currently defined:

Table 33. PSP-to-BIOS Mailbox Commands

Command	Value	Description/Purpose
MboxPspCmdSpiGetAttrib	0x81	Obsolete
MboxPspCmdSpiSetAttrib	0x82	Obsolete
MboxPspCmdSpiGetBlockSize	0x83	Get SPI-ROM block size
MboxPspCmdSpiReadFV	0x84	Read PSP NVRAM firmware volume
MboxPspCmdSpiWriteFV	0x85	Write SP NVRAM firmware volume
MboxPspCmdSpiEraseFV	0x86	Erase PSP NVRAM firmware volume

7.2.1 PSP to BIOS Mailbox Commands

To request BIOS services the PSP firmware builds mailbox commands in SMM space and generates an SMI. PSP firmware uses information per the BIOS mailbox command MboxBiosCmdSmmInfo and uses the PSP SMM area and SMM trigger mechanism to trigger BIOS services. The structure of the BIOS mailbox interface is defined below:

```

typedef struct {
    UINT32 CheckSumValue:8;    ///< 8bits Checksum Value of MBOX_BUFFER
    UINT32 CheckSumEn:1;      ///< Switch for Enable/Disable Checksum
    UINT32 Reserved:22;       ///< Reserved
    UINT32 CommandReady:1;    ///< Flag used specify P2C SMI command is ready to handle
                                a new command
} SMI_MBOX_STATUS;

typedef struct {
    volatile MBOX_COMMAND      MboxCmd;          ///< Mbox Command 32 bit wide
    volatile SMI_MBOX_STATUS   MboxSts;          ///< Mbox status 32 bit wide
    MBOX_BUFFER                Buffer;            ///< Mailbox buffer right after

```



```
} BIOS_MBOX;
```

During cold boot path BIOS SMM driver allocates this region with enough size and informs PSP firmware about this region through MboxBiosCmdSmmInfo mailbox command. PSP firmware must never use SMM space beyond the allocated SMM space. When BIOS SMM code is ready to serve PSP SMM requests it sets the CommandReady bit in MboxSts register.

When PSP firmware requires sending command to BIOS, it will 1st check if SMI issue condition matches via comparing the register value as MboxBiosCmdSmmInfo reported. PSP firmware next sets MboxCmd with a specific mailbox command, and fill the MBOX_BUFFER that PSP needs BIOS to service. If checksum mechanism enabled, PSP will also set CheckSumEn bit, and calculate the 8bits Checksum Value of MBOX_BUFFER. After the BIOS_MBOX structure is setup by PSP firmware it triggers an SMI to the x86 core.

After the SMI is issued, the AmdPspP2Cmbox SMM driver handles the SMI. It will 1st set CommandReady to 0 to identifying the command is processing, then check the command's validity via ensure the buffer address is reside in the allocate memory region; the command ID is valid and calculated the checksum if checksum mechanism is enabled. The handle routine will exit with error status set if any condition doesn't matches. The handler then parses the structure and invokes the appropriate routine to service the PSP request. After the completion of the request, the BIOS SMM driver updates the MboxSts register to reflect completion status and clears the MboxCmd location, set CommandReady to indicate completion of the PSP command, and calculated the checksum of MBOX_BUFFER again, if checksum is enabled.

The structure of command/response buffer at offset 8 of BIOS_MAILBOX will change based on PSP command. The generic format of command/response buffer is similar to the BIOS to PSP command and is listed below:

```
typedef struct {
    UINT32    TotalSize;
    UINT32    Status;
} MBOX_BUFFER_HEADER;

typedef struct {
    MBOX_BUFFER_HEADER    Header;
    COMMAND_SPECIFIC_BUFFER    Buffer;
} MBOX_COMMAND_RESPONSE_BUFFER;
```

Each command/response buffer starts with a standard mailbox buffer header. The field TotalSize is set to size of command/response buffer. After the PSP command is serviced the field Status is updated by the BIOS SMM driver to reflect the status of the command requested by PSP firmware. The standard mailbox buffer header is followed by a command specific buffer that is different per each mailbox command.

PSP firmware should never attempt to write/erase SPI space beyond PSP NVRAM region. BIOS SMM code must reject any such access.

To support multiple PSP NV region access, TargetNvId field is added to the all commands as below:

```
typedef enum {
    SMI_TARGET_NVRAM = 0,
    SMI_TARGET_TOKEN_UNLOCK = 1,
    SMI_TARGET_VM_GUARD = 2,
    SMI_TARGET_NO_USE = 0xFE,
    SMI_TARGET_END = 0xFF,
} SMI_TARGET_ID;
```

7.2.1.1 MboxPspCmdSpiGetBlockSize (MboxCmd = 0x083)

PSP firmware sends this command to request

EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL.GetBlockSize() service on PSP NVRAM region in SPI space. PSP firmware requires this information to detect the size of each block in PSP NVRAM region.

Note:

```
typedef struct {
    UINT64 TargetNvId;           ///< Target NV ID
    UINT64 Lba;                 ///< starting LBA
    UINT64 BlockSize;           ///< Block size of each Lba
    UINT64 NumberOfBlocks;      ///< Total number of blocks
} SPI_INFO_REQ;
```

BIOS returns the Lba as (zero), BlockSize and total blocks of PSP NVRAM region.

7.2.1.2 MboxPspCmdSpiReadFV (MboxCmd = 0x084)

PSP firmware sends this command to request

EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL.Read() service on PSP NVRAM region in SPI space.

PSP firmware may use this command to read its data from PSP NVRAM space. Generally PSP firmware will directly perform the SPI read and not use this command. The command-specific structure for this command is below

```
typedef struct {
    UINT64 TargetNvId;           ///< Target NV ID
    UINT64 Lba;                 ///< starting LBA
    UINT64 Offset;              ///< Offset in LBA
    UINT64 NumByte;             ///< Total byte to read
    UINT8 Buffer[1];            ///< Buffer to read the data
} SPI_RW_REQ;
```

Lba is the logical block address in NVRAM that PSP firmware requests to read. Offset is the specific location within the Lba block. NumByte field reflects the number of bytes PSP firmware needs to read. BIOS SMM code is expected to read the requested NVRAM offset and update the SMM space starting at offset Buffer.

7.2.1.3 MboxPspCmdSpiWriteFV (MboxCmd = 0x085)

PSP firmware sends this command to request

EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL.Write() service on PSP NVRAM region in SPI space. PSP firmware may use this command to write to PSP NVRAM space. To protect the PSP NVRAM region BIOS SMM code is expected to lock this region unless PSP made this write request. The structure for this command is the same as above SPI_RW_REQ. In this command, BIOS reads the SMM location starting at Buffer offset of the structure and writes this buffer content to the appropriate SPI location.

7.2.1.4 MboxPspCmdSpiEraseFV (MboxCmd = 0x086)

PSP firmware sends this command to request

EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL.Erase() service on PSP NVRAM region in SPI space.

BIOS SMM driver erases the SPI space in response to this command. The request buffer for this command specific structure is defined below:

```

typedef struct {
    UINT64  TargetNvId;           ///< Target NV ID
    UINT64  Lba;                  ///< starting LBA
    UINT64  NumberOfBlocks;       ///< Total number of blocks
} SPI_ERASE_REQ;

```

In response to this command, BIOS erases the SPI space per Lba & NumberofBlocks as defined in the structure.

7.3 MP0_P2C_MSG reserved for X86 and PSP information sharing

Beside of BIOS to PSP mailbox and PSP to BIOS mailbox, there are also some MP0_P2C_MSG registers reserved for information sharing. Details as below table:

MP0_P2C_MSG Register	Description
MP0_P2C_MSG_1	Error code for Platform secure boot: BL_ERROR_LOAD_OEMSIGNING = 0x08, // OEM Key is not present BL_ERROR_RSAPSS_OEMSIGNING = 0x1D, // OEM Key is not valid BL_ERROR_KEYUSAGE_BIOS_PEI = 0x2C, // OEM Key usage is wrong BL_ERROR_LOAD_BIOS_PEI = 0x09, // BIOS PEI is not present BL_ERROR_RSAPSS_BIOS_PEI = 0x1E, // BIOS PEI is not valid
MP0_P2C_MSG_2	Reserved for native mode secure debug unlock without JTAG connection
MP0_P2C_MSG_3	Reserved for native mode secure debug unlock without JTAG connection

Chapter 8 Platform BIOS Requirements for PSP Implementation

The below BIOS requirement applies when platform BIOS supports Platform Secure Boot and PSP firmware TPM feature.

- Platform BIOS MUST reserve some space in SPI storage for PSP components. This includes PSP firmware, SMU Firmware, PSP NVRAM data, various keys, and other firmware components in the PSP Directory [IBV|OEM] .
 - Platform BIOS must include PSP Directory and BIOS Directory as outlined in section 4.1 on page 33. For more information about the tool used to build PSP Directory please refer to Appendix A, “PSP Directory Structure” on page 103.
 - AGESA PI package must include all PSP and SMU binaries
 - Platform BIOS must include PSP directory. All PSP appropriate entries defined in Table 5 on page 43, must be included in PSP Directory.[IBV|OEM]
 - Public portion of OEM signing key MUST be signed by AMD and both OEM signed key and AMD public key MUST be present in SPI image and also locatable via PSP directory. [IBV|OEM]
- Platform BIOS MUST set up PSP code and data pages in SPI space such that the PSP code and data page can be independently erased/updated without affecting the rest of the BIOS components in SPI space.[IBV|OEM]
- CMOS shadow region (IO_Index72/73[0x5:0x7]) is used for ABL CMOS power loss check, Platform BIOS need to reserve and manage these CMOS bytes
- BIOS MUST support single binary that supports both secure/non-secure boot and resume environment.[IBV|OEM]
- BIOS MUST include necessary support to ensure the trust chain is maintained from x86 BIOS reset to OS handoff. [IBV|OEM]
 - If Platform Secure Boot is enabled or enforced, IBV|OEM MUST sign the BIOS RTM code (PEI Volume or SEC volume) using private key based on RSASSA-PSS signing scheme. The private portion of IBV|OEM signing key MUST be protected per IBV|OEM process choice. If

- IBV|OEM private keys are compromised the overall Platform Secure Boot feature can be completely compromised across all systems. [IBV|OEM]
- Platform BIOS MUST build PSP Directory that provide location information of BIOS_RTM location, signed BIOS RTM location and signed BIOS public key in PSP directory.[IBV|OEM]
 - If BIOS RTM volume has only SEC code then it MUST authenticate the PEI volume before handing off control to PEI core.[IBV|OEM]
 - BIOS PEI volume must authenticate DXE and other firmware volume during cold boot and resume before executing BIOS drivers from these volumes to protect the integrity of trust chain. [IBV|OEM]
 - Platform BIOS must authenticate external software component such as external Option Rom and OS Boot loader as defined in secure boot section of UEFI specification.
 - Platform BIOS MUST send PSP mailbox commands as outlined in the PSP-BIOS mailbox section. [AGESA/OEM|IBV]
 - AGESA PSP/fTPM PEI, DXE and SMM driver MUST send the all BIOS-PSP mailbox command to PSP at appropriate point during boot. Similarly AGESA PSP/fTPM driver should handle PSP-BIOS mailbox command at runtime[AGESA]
 - Platform BIOS code MUST ensure these AGESA drivers are appropriately loaded and protected.[IBV|OEM]
 - PSP MMIO region MUST be always accessible for BIOS-PSP mailbox communication. [AGESA/IBV|OEM]
 - Platform BIOS MUST provide an SMM interface to allow PSP data to be saved in SPI space. This library interface as defined in section 10.3.2 in SMM space that will be used to update PSP NVRAM data.[IBV|OEM]
 - Platform BIOS MUST ensures this interface is inaccessible outside SMM. Also the PSP region of SPI MUST remain write protected outside this interface use such that NVRAM cannot be tempered [IBV|OEM]
 - AGESA P2Cmbox SMM driver MUST use this library interface in SMM space to locate the storage protocol [AGESA]
 - BIOS MUST reserve 4K SMM memory for BIOS-PSP runtime communication.[AGESA]
 - AGESA PspP2Cmbox SMM driver MUST reserve region for BIOS-PSP communication [AGESA]
 - Platform BIOS must provide SMM kernel services to allow SMM memory allocation with in SMM space.[IBV|OEM]
 - Platform BIOS MUST ensure the SMM environment is protected
 - SMM region MUST be locked before exiting the BIOS boot environment. The SMM code must remain protected such that non-authenticated code cannot access SMM data or code area. Some additional note, the SMM lock need be done after PspDxe driver save CPU core context through the specific MSR, if SMM region need to access during the SMM resume path. [IBV|OEM]

- Platform BIOS MUST ensure the SMI source used by PSP is configured properly for SMM to be non-blocking during runtime.[AGESA]
 - AGESA PSP driver MUST install appropriate SMM handle to service SMI events (FakeSts0/ Software SMI) from PSP firmware. AGESA PSP driver must also provide this information to PSP firmware via mailbox interface.
 - Platform BIOS MUST ensure the same SMI mechanism is not used by other SMM component. Platform BIOS must also ensure the chipset registers remain configured to trigger SMI at run time. [IBV|OEM]
- BIOS MUST adequately reserve system memory during boot to appropriately set up stack/heap for DRAM ready x86 SMM resume path on a secure PSP part.[AGESA/ IBV|OEM]
 - AMD AGESA SMM driver MUST reserve the SMM location [AGESA]
 - Platform driver MUST produce AMD_PSP_PLATFORM_PROTOCOL instance that AMD AGESA SMM driver can use to reserve SMM space.[IBV|OEM]
 - If platform BIOS build a separate resume volume as outlined in section 5.2, on page 87 the platform BIOS must reserve this memory region to protect against any OS use. Also during resume platform BIOS MUST ensure this resume volume in DRAM has not been tempered.[IBV|OEM]
- BIOS MUST support Windows 8 Secure Boot requirements as specified in Microsoft's certification requirements.[IBV|OEM]
- BIOS MUST support Windows 8 firmware TPM interfaces as specified in Microsoft's whitepaper "Trusted execution environment ACPI profile". This includes reserving part of system memory that will be used by PSP firmware as a TPM Command and Response Buffer.[IBV|OEM]
 - BIOS MUST send fTPM command after memory is available.
 - BIOS MUST protect the PSP code and data area in the SPI space. This area must be write-protected via SMM or physical lock so that only BIOS SMM code can update this region.[IBV|OEM]
- BIOS flash utility MUST ensure PSP firmware is properly updated and also the SPI region for PSP firmware is properly protected during regular boot.[IBV|OEM]
 - During flash update platform BIOS must ensure the integrity of contents of PSP Directory as well as various binaries referenced by PSP directory. PSP firmware searches for PSP Directory in multiple locations as defined in Section 4.1.4 on page 42. BIOS vendor MAY use these other location as backup location. PSP firmware looks for PSP directory in the order as defined in Table 1 on page 17.
- BIOS must support UEFI TPM protocol for TPM devices. For ex. "TCG EFI Protocol Specification" for onboard discrete TPM1.2 device and EFI_TREE_PROTOCOL [9] for TPM2.0 devices.[IBV|OEM]
- When a platform BIOS (such as Larne reference board) supports both discrete and firmware TPM for development and validation purposes ONLY, the platform BIOS MUST support both kind of TPM stack (i.e., TPM1.2/TPM2.0 as well as dTPM/fTPM) in the BIOS ROM image. During cold boot the platform BIOS MUST load appropriate TPM driver stack based on user configuration

(such as BIOS setup option) option. On AMD reference validation board, the default TPM selection MUST be internal firmware TPM use.[IBV|OEM]

- When fTPM is enabled, the PSP NV storage area saved in boot media needs be preserved for BIOS update.
- Platform BIOS MAY use PSP entry type 0x80–0xFF for OEM or IBV specific proprietary use. PSP Entry Type from 0x00 to 0x7F is reserved for AMD use.
- Platform BIOS MAY use PSP Hardware Cryptographic accelerator to improve the boot time associated with cryptography operation such as AES, SHA, RSA, ECC, RNG.[AGESA/IBV|OEM]
 - AMD AGESA MAY provide the library for the base CCP operation mentioned above [AGESA]
 - IBV|OEM CCP library MAY use the AMD AGESA CCP library to make use of PSP CCP.
- IBV BIOS MUST ensure SPI decode range in FCH controller remain properly programmed for PSP firmware to access other PSP component from SPI space.

Chapter 9 AMD AGESA™ PI Interface Note

AGESA™ PI source code package will include several libraries and PSP UEFI drivers to communicate with PSP firmware. The following is a list of drivers and libraries that will be included.

9.1 Library

- PspMbox library includes the library functions for BIOS to send the mailbox commands to PSP. Appendix B provide list of those mailbox command.
- PspfTpm library includes the library function for BIOS to send the fTPM command to PSP.

9.2 Drivers

- PspPei — This PEI driver configures the PSP PCIe® configuration MMIO space and enables BIOS-PSP communication. In addition it sends mailbox command to PSP firmware when the memory is ready for use.
- PspDxe — This DXE registers for ReadyToBoot event to send MboxBiosCmdBootDone command to PSP FW, and trigger SMI for PSPSmm driver to perform actions (discussed next) on ReadyToBoot event. In addition the RdRand related register setting will be initialized by this driver.
- PspSmm — This is a SMM driver and performs several tasks.
 1. Register sleep trap SMI, and send MboxBiosCmdSxInfo command to the PSP firmware inside of the handler.
 2. Handle the resume path. It performs CPU MSR write at end of BIOS POST (i.e., ReadyToBoot event, to save the CPU context for S3 resume). As part of MSR write the driver setup the rip value of resume vector so that x86 reset on S3 resume will fetch and run SMM code in PSP driver
 3. During resume from sleep state the driver calls various callback functions of other driver to allow chipset restore (discussed later) by other drivers. The driver produces protocol PSP_RESUME_SERVICE_PROTOCOL that allow other drivers to register the callback services that will be services during resume time.
 4. After the SMM callbacks are appropriately handled the PspSmm driver patches the CPU SMM save area and resume out of SMM mode to BIOS SEC code outside SMM. The driver uses the IBV resume preference information by locating and consuming information in PSP_PLATFORM_PROTOCOL (defined later) and sets up SMM save area for GDT table, stack etc based on this protocol. When CPU resume out of SMM state it use the SMM save area context to jump to SEC/PEI code per the SMM state information. After PSP hand off

control to IBV SEC code the rest of PEI driver in resume PEI volume perform rest of x86 restore and handoff control to OS.

- The general flow of PSP SMM driver is below
 - When BIOS configuration is complete (i.e., ReadyToBootEvent) the driver perform CPU MSR write to build cpu conext in fenced C6 area for resume.
 - The driver use the IBV produced protocol PSP_PLATFORM_PROTOCOL to find relevant information during S3 resume path (discussed next) to handoff control to BIOS resume SEC code.
 - The driver produces protocol PSP_RESUME_SERVICE_PROTOCOL for other BIOS SMM driver to register their callback function during resume time. Other SMM drivers can locate this protocol and use Register service of this protocol to add callback functions. These registered callback function will be invoked by PSP SMM driver during resume and these callback function can perform additional restore operation of other devices; After invoking all the callbacks the PSP SMM driver hand off control to BIOS resume vector (outside SMM space) in SEC mode.
 - In addition PSP SMM driver send mailbox command to PSP just before system transition to sleep state. The PSP firmware performs any last minute operation before system transition to S3.
- PspP2Cmbox — Establish the PSP to CPU mailbox communication and send MboxBiosCmdSmmInfo command.
 - The driver allocates SMM memory and informs PSP firmware though mailbox command regarding the location of this area and mechanism to trigger SMI.
 - Hook with Smm core services to handle the SMI that will be triggered by PSP firmware to request SPI storage services. The IBV is expected to prepare SPI SMM library that produces EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL protocol liked interface. The PSP SMM driver locates and uses this Library to handle PSP SMM request for storage.
- fTpmPei

Assign memory for resource (1M * 2) for TPM command and response buffer. Publish TPM TIS PPI.
- fTpmDxe

Assign memory for resource (1M * 2) for TPM command and response buffer. Publish TPM TIS Protocol
- Addendum\fTPMAcpi

The driver provides the sample codes to publish TPM2 ACPI SSDT table and associated TPM2 ASL codes. For details of Microsoft TPM2 ACPI requirement, please refer to Section 10.3 on page 121.
- Addendum\NewRsmSampleDriver

This driver provides the sample codes for callback handler in SMM Resume path by using AMD_PSP_RESUME_SERVICE_PROTOCOL.

- Addendum\PspPlatformDriver

The driver provides the sample codes to implement AMD_PSP_PLATFORM_PROTOCOL

9.3 UEFI PPI/Protocol Consumed/Produced by PSP Drivers

9.3.1 PSP_FTPM_PPI

Summary

The PSP FTPM PPI provide the functionality including get fTPM info;

Send TPM command; Get TPM command's response; Execute TPM command (including send and get response). This PPI will be published only when fTPM interface is available, IBV can use this PPI as a dependency before sending any TPM command, in case fTPM library is selected instead of the PPI.

GUID

```
#define PSP_FTPM_PPI_GUID \
    {0x91774185, 0xf72d, 0x467e, 0x93, 0x39, 0xe0, 0x8, 0xdb, 0xae, 0xe, 0x14}
```

Protocol Interface Structure

```
typedef struct _PSP_FTPM_PPI {
    FTPM_EXECUTE Execute;           ///< Execute TPM command,
    include send & get response
        FTPM_CHECK_STATUS CheckStatus;    ///< Check TPM Status
        FTPM_SEND_COMMAND SendCommand;    ///< Send TPM command
        FTPM_GET_RESPONSE GetResponse;    ///< Get Last TPM command
    response
} PSP_fTPM_PPI;
```

Parameter

- **Execute**

Execute a TPM command

```
typedef
EFI_STATUS
(EFIAPI *fTPM_EXECUTE) (
    IN      PSP_fTPM_PPI    *This,
    IN      VOID             *CommandBuffer,
    IN      UINTN            CommandSize,
    IN OUT VOID             *ResponseBuffer,
    IN OUT UINTN            *ResponseSize
);
```

This

Point to the PPI instance

`CommandBuffer`

Point to the TPM command buffer

`CommandSize`

Size of the TPM command buffer

`ResponseBuffer`

Point to the TPM response buffer

`ResponseSize`

Size of the TPM response buffer

- `CheckStatus`

GET TPM related Info, currently only fTPM support capability can't got from the returning status, `EFI_SUCCESS` Identify fTPM function supported or else unsupported.

```
typedef
EFI_STATUS
(EFIAPI *fTPM_CHECK_STATUS) (
    IN      PSP_fTPM_PPI          *This,
    IN OUT UINTN                  *fTPMStatus
);
```

`This`

Point to the PPI instance

`fTPMStatus`

Unused Currently

- `SendCommand`

Send a TPM command, refers `ExecuteCommand` for parameter's definition.

- `GetResponse`

Get a TPM command's response, refers `ExecuteCommand` for parameter's definition.

9.3.2 PSP fTPM Protocol

Summary

The PSP fTPM Protocol provide the functionality including get fTPM info;

Send TPM command; Get TPM command's response; Execute TPM command (Including send and get response)

GUID

```
#define fTPM_PROTOCOL_GUID \
{ 0xac234e04, 0xb036, 0x476c, 0x91, 0x66, 0xbe, 0x47, 0x52, 0xa0, 0x95, 0x9 }
```

Protocol Interface structure

Same as PSP_FTPM_PPI

9.3.3 AMD_PSP_PLATFORM_PROTOCOL**Summary**

No need for ARM ISA

The PSP Platform protocol provides the customization information to PSP SMM driver. The information in this protocol is used by PSP SMM driver to prepare SEC handoff mechanism during resume path.

GUID

```
#define AMD_PSP_PLATFORM_PROTOCOL_GUID \
    { 0xccf14a29, 0x37e0, 0x48ad, { 0x90, 0x5, 0x1f, 0x89, 0x62, 0x2f, 0xb7, 0x98 } }
```

Protocol Interface Structure

```
typedef struct _PSP_PLATFORM_PROTOCOL {
    BOOLEAN                CpuContextResumeEnable;
    UINT8                  SwSmiCmdtoBuildContext;

    UINT32                  BspStackSize;
    UINT32                  ApStackSize;
    RSM_HANDOFF_INFO        *RsmHandOffInfo;
} PSP_PLATFORM_PROTOCOL;
```

Parameter

CpuContextResumeEnable

Indicate platform BIOS supports SMM based resume path by saving CPU context during boot. If set to TRUE the PSP SMM driver write to CPU MSR to build the CPU context. In this case the resume from S3 will start x86 core in SMM mode.

SwSmiCmdtoBuildContext:

This is the software SMI value that PSP SMM driver will use to prepare CPU Context. The PSP DXE driver will trigger this software SMI when CPU context is properly built (i.e., all the MSR and CPU microcode is properly configured). The PSP SMM driver will register for this software SMI and on this SMI event it will write to CPU MSR on all cores to save the context of each CPU core.

BspStackSize

PSP SMM drive uses this information to calculate the stack pointer of BSP and AP in SMM save area when system resumes from sleep state. The stack will be allocated per PspDramRsdLength and Bsp Stack is built from that pool and stack length of BSP is calculated per this parameter. AP stack will start right after BspStackSize.

ApStackSize

PSP SMM driver uses this information to calculate the stack pointer of each AP in SMM save area when system resumes from sleep state.

RsmHandOffInfo

This structure is used to convey SEC handoff information to PSP SMM driver. The PSP SMM driver use this structure to patch the GDT Table, code and data selector, RIP of SEC code in the SMM save area. When PSP SMM driver rsm out of SMM mode later the SEC code will get control per this information (i.e., GDT table will point to this structure, code and data selector will point to selector value per this structure and SEC code will run code per RsmEntry point).

Related Parameter

```
typedef struct {
    UINT32  GdtOffset;           // GDT table offset for RSM
    UINT16  CodeSelector;        // CODE Segment Selector
    UINT16  DataSelector;        // DATA Segment Selector
    UINT32  RsmEntryPoint;       // IP Address after executing rsm command
    UINT32  EdxResumeSignature;  // Value keep in EDX after executing rsm command
} RSM_HANDOFF_INFO;
```

GDTOffset

Location of GDT table in resume PEI volume. If there is single PEI volume for cold/resume path this location will point to GDT table in SPI rom area.

CodeSelector

The selector value of code segment. CS offset of SMM save area will be updated with this value.

DataSelector

The selector value of data segment

RsmEntryPoint

SEC entry point during resume. The SMM save area at offset rip will be updated with this value. When PSP SMM driver exit out of SMM mode, to platform SEC code in PEI volume, the x86 will rsm to platform SEC code will resume execution from this address

EdxResumeSignature

GDTOffset

Location of GDT table in resume PEI volume. If there is single PEI volume for cold/resume path this location will point to GDT table in SPI rom area.

CodeSelector

The selector value of code segment. CS offset of SMM save area will be updated with this value.

DataSelector

The selector value of data segment

RsmEntryPoint

SEC entry point during resume. The SMM save area at offset rip will be updated with this value. When PSP SMM driver exit out of SMM mode, to platform SEC code in PEI volume, the x86 will rsm to platform SEC code will resume execution from this address

EdxResumeSignature

If there are additional parameter that SEC code needs to use this parameter can be used for such purpose. The value in this field will be updated in edx register offset location of SMM save area. When SEC code gets control from PSP SMM driver the EDX register will reflect this value. This field can specify a signature value or the location of any context that SEC code need to use during resume path.

9.3.4 AMD_PSP_RESUME_SERVICE_PROTOCOL

Summary

Removed in ARM ISA

Install/Uninstall callback services to be dispatched by PSP SMM driver when resuming from resume path.

GUID

```
#define AMD_PSP_RESUME_SERVICE_PROTOCOL_GUID \
    {0x49e7712, 0xd66a, 0x4e0d, 0xb0, 0x24, 0x7, 0x59, 0x40, 0x14, 0x3e, 0x42}
```

Protocol Interface Structure

```
typedef struct _PSP_RESUME_SERVICE_PROTOCOL {
    PSP_RESUME_REGISTER    Register;
    PSP_RESUME_UNREGISTER  UnRegister;
} PSP_RESUME_SERVICE_PROTOCOL ;
```

Parameter

- Register

Install a callback function to be dispatched when resuming from sleep or connected standby

```
typedef EFI_STATUS (EFI_API *PSP_RESUME_REGISTER) (
    IN        PSP_RESUME_SERVICE_PROTOCOL    *This,
    IN        PSP_RESUME_CALLBACK            CallbackFunction,
    IN OUT    VOID                            *Context,
    IN        UINTN                           CallbackPriority,
    OUT       EFI_HANDLE                       *DispatchHandle
);
```

CallbackFunction

Address of callback function that is registered to be called when resuming from sleep or connected standby

Context

Saved context of callback function

CallbackPriority

Priority of the callback. This index will be used to decide the order of callback (0–low, 0xffffffff high)

DispatchHandle

The associated handle for this callback. This handle can be used to unregister the callback function.

- **UnRegister**

Remove a callback function from the callback list that was earlier registered.

```
typedef EFI_STATUS (EFI_API *PSP_RESUME_UNREGISTER) (
    IN      PSP_RESUME_SERVICE_PROTOCOL    *This,
    IN      EFI_HANDLE                     DispatchHandle
);
```

DispatchHandle

Handle associated with the callback that should be unregister.

Related Parameter

```
typedef enum {
    ResumeFromConnectedStandby = 0x01, // Resume from Connected Standby
    ResumeFromS3                = 0x02  // Resume from S3 sleep
} RESUME_TYPE;
```

ResumeType enum field provide information to callback handler the type of resume path (i.e., if the callback is invoked on resume from S3 path or resume from connected standby path).

```
typedef EFI_STATUS (EFI_API *PSP_RESUME_CALLBACK) (
    IN RESUME_TYPE    ResumeType,
    IN VOID           *Context
);
```


Chapter 10 Standards

The PSP and software components must comply with many standards including the following:

10.1 UEFI 2.3.1c Chapter 27 Secure Boot

Affected component(s): System BIOS

<http://www.uefi.org/specs/>

10.2 Microsoft[®] Trusted Execution Environment UEFI Protocol

Affected component(s): System BIOS, Firmware TPM, PSP OS

<http://msdn.microsoft.com/en-us/library/windows/hardware/jj923068.aspx>

10.3 Microsoft[®] Trusted Execution Environment ACPI Profile

Affected component(s): System BIOS, Firmware TPM, PSP OS

<http://msdn.microsoft.com/en-us/library/windows/hardware/jj923067.aspx>

10.4 AMD PSP 1.0 Software Architecture Design Document

Affected component(s): System BIOS, Firmware TPM, PSP OS

Appendix A PSP S5 Boot Flow

When SoC is powered-up and reset is de-asserted SMU On-chip Boot ROM code starts executing and early fuse loading takes place. At this point both PSP microcontroller (A5) and Host CPU (x86) are both in reset. Subsequently host x86 BSP core and PSP (A5) resets are both de-asserted.

A.1 Boot Flow — S5 Cold Boot

Please refer to Section 3.2.1 on page 25 and 3.3 on page 31 for additional details.

When the SOC-15 is powered up, there is only one processor running, the Cortex-A5 in MP0 block. All Zen x86 complexes and cores are in halted/reset state. All factory fuses are loaded by on-chip BootROM, which then validates and loads the off-chip Bootloader from SPIROM to SRAM to begin boot-up ending in launching BIOS.

Note that the BootROM will ensure that both host x86 BSP core and PSP (A5) resets are both de-asserted.

Figure 14 below provides the operation flow from SOC power-on for S5 boot.

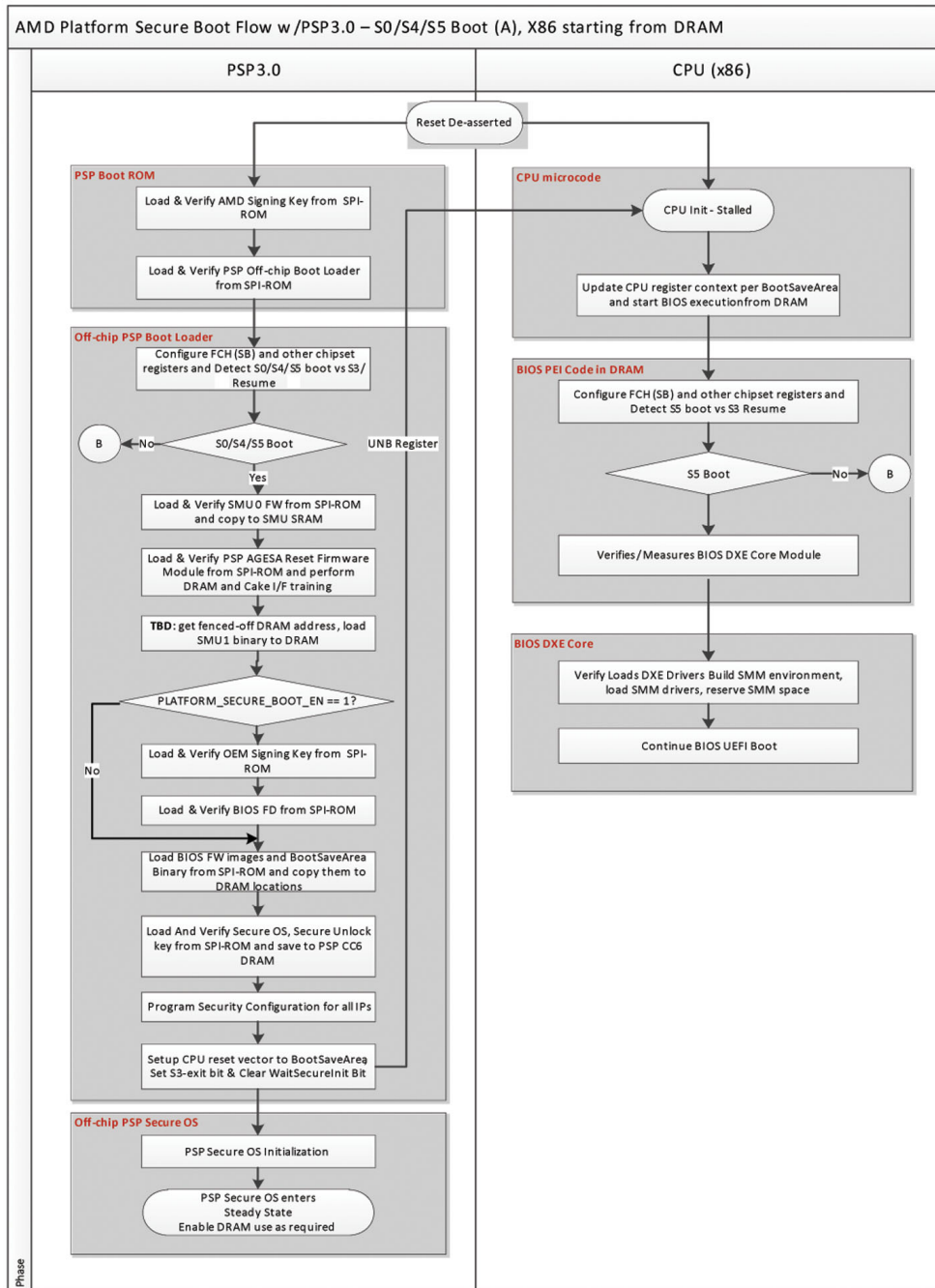


Figure 12. AMD Family 17h Models 00h–0Fh Processor Boot Flow — S5 Cold Boot

Appendix B BuildPspDirectory Tool Version 4.x

BuildPspDirectory Version 4.x is available to achieve the following goals:

- Need to support different configuration file
- Need to support different address mode, Physical Address for SPI, Relative Address for eMMC and UFS
- Need to support BIOS Directory, with Type field updated, and Destination field added.
- Need to support multiple instance for same entry type

Also design goal for BuildPspDirectory configuration file are as below:

- Should be both human and machine readable;
- Should be ease for extend
- Better following an industry standard.

Due to above considerations, the PSP Directory configuration file format has a big change from previous self-defined plain text to generic XML format. Also the command line parameter has a slight change to support output directory, and allow specify the input XML file.

B.1 PSP Directory Configure File Format

The PSP Directory Configuration is following XML format.

B.1.1 Node <DIRS>

The root node of the configure file, a **DIRS** tag can consist of one BIOS_DIR or PSP_DIR node, or two at the same time

Valid Values: 0, 1, 2, 3. Tool fills physical address if AddressMode is cleared to 0. Tool fills relative address to the entire BIOS image if AddressMode is set to 1.

If AddressMode is set to 2, the tool fills relative address based on each directory base. For example:

- If L2A PSP directory has a base address 0x140000, the L2A BIOS directory has base address 0x160000.
- If a point entry in the PSP directory has address 0x2000, then the real binary offset is $0x140000 + 0x2000 = 0x142000$.
- If a point entry in the BIOS directory has address 0x22000, then the real binary offset is $0x160000 + 0x22000 = 0x182000$.
- If PSP_DIR has base address 0x140000, and a point entry has address 0x2000, then the real binary offset is $0x140000 + 0x2000 = 0x142000$.

If AddressMode is set to 3, the tool fills relative address based on each slot (for PSP directory, it is equal to AddressMode2; for BIOS directory, the address is relative to the corresponding PSP directory offset). For example:

- If L2A PSP directory has base address 0x140000, the L2A BIOS directory has base address 0x160000.
- If a point entry in PSP directory has address 0x2000, then the real binary offset is $0x140000 + 0x2000 = 0x142000$.
- If a point entry in BIOS directory has address 0x22000, then the real binary offset is $0x140000 + 0x22000 = 0x162000$.

If not specified, the default address Mode is the physical address.
(e.g., "<DIRS AddressMode='1'>" sets the address mode to relative.)

B.1.2 Node <PSP_DIR>

Specify the entries belong to PSP Directory, these pair attributes Base and Size are used to reserve a region in the flash to hold the directory header as well as the files to be embed, if these attributes have been defined, the tool will automatically calculate the address for child "IMAGE_ENTRY" node, or else the "Address" should be explicitly declared in every child "IMAGE_ENTRY"

1. Attribute: Base, Headerbase (Optional)

Define the Base of reserved region for PSP Directory used to hold the FW image belong to it.. It is using the relative address of binary.

If "Headerbase" is specified, the directory header structure will place at the region which "Headerbase" point to, or else, the directory header structure and firmware images will place continual which "Base" point to.

With "Headerbase" declared, it allow to separate the PSP Directory header and the region used to hold the firmware. (e.g., <PSP_DIR Level="1" **HeaderBase**="0x146000" **Base**="0x1146000" Size="0x100000" SpiBlockSize="0x1000">)

In this example, the PSP directory header structure is placed at Address 0x146000, the FW belonging to the PSP Directory is placed at 0x1146000

It implies that the engineer wants to put the PSP directory header in lower 16M of SPI, and PSP FW in the higher 16M.

2. Attribute: Size (Optional)

Define the Size of reserved region for PSP Directory

3. Attribute: Copy (Optional)

This attribute is mainly for A/B recovery use.

If Copy is set to 0x1, the directory will copy all information other than **Base** attribute from the previous Level 2 directory in the same configure file.

If no level 2 directory is found, the tool generates an error.

All attributes of the directory other than **Base** and all nodes are ignored.

The tool picks up the **Base** attribute to set the directory base address.

For example, <PSP_DIR AddressMode="0x2" Base="0x870000" Level="0x2" Size="0x12a000" SpiBlockSize="0x1000" Copy="0x1">.

4. Attribute: AddressMode (Optional)

AddressMode can be set individually for each directory. This overwrites the AddressMode in **DIRS** tag. If AddressMode is not specified, it inherits from the **DIRS** tag.

5. Node <IMAGE_ENTRY>

Specify the entry need tool patch/embed the file to final image

a. Attribute: Type (Required)

Specify the type as defined in Table 8, "AMD Family 17h and 19h Processor PSP Directory Type Encodings" on page 44 .

b. Attribute: File (Optional)

Specify the file want the tool to embed or patched. The tool will embed the file to the address by auto calculation if Address not explicit defined, or to the address "Address" attribute defined. This attribute is required if "Size" attribute is not set.

c. Attribute: Size (Optional)

Size can be used to overridden the real size of file, or reserve a region of given size in the PSP directory area, should be set large enough to hold the file. It can be used in the case where the file is not presented when build the Directory header by using "bd" command line parameter. For example, RTM signature is generated by doing the signature of catenated PSP directory header and RTM binary, which is not existed when execute "bd" command. In this case, Size attribute can be used to reserve a size of region during executing "bd" command.

For example, the Sample entry for "Size" Overridden

```
<IMAGE_ENTRY Type="0x00" File="RTMSignature.bin" Size=" 0x100"/>
```

This attribute is required if "File" attribute is not set.

d. Attribute: Address (Optional)

Explicit define the address for tool to embed. It uses the relative address of binary.

e. Attribute: RomId (Optional)

Explicit declare which SPI device PSP load from, 0: the SPI device select by SPI_CS1_L, 1: the SPI device select by SPI_CS2_L.

Currently there are two typical user cases:

1. Two SPI images, all runtime update images are required to to put in the SPI_CS2_L device.

e.g.,

1. For fTPM feature

```
<!-- PSP Non Volatile data -->
<POINT_ENTRY Type="0x04" Address = "0x41000" Size = "0x20000"
RomId="1"/>
```

2. For token unlock feature

```
<!-- Token unlock Data -->

<IMAGE_ENTRY Type="0x22" File="SecureEmptyToken.bin" RomId="1"/>
```

3. Input for ABL

```
<!-- Agesa PSP Customization Block (PSP Agesa variable) NVRAM (APCB)-
->
```

```
<IMAGE_ENTRY Type="0x60" File="APCB_RV_D4.bin" Size="0x2000"
RomId="1"/>
```

4. NV to save ABL s3 data

```
<!-- APOB NVRAM COPY-->
<IMAGE_ENTRY Type="0x63" File="APOB_NV_RV.bin" RomId="1"/>
```

2. BIOS reset image is placed in SPI_CS2_L device

e.g.,

```
<!--Reset Image is placed at ROM 1-->
```



```

<POINT_ENTRY Address="0x750000L" Destination="0x09A00000"
Size="0x200000" Type="0x62" Instance="0x00" RomId="1">

  <TypeAttrib Copy="0x1" ReadOnly="0x0" RegionType="0x0" ResetImage="0x1"
    Compressed="0x1"/>

</POINT_ENTRY>

<POINT_ENTRY Address="0x750000L" Destination="0x09C00000"
Size="0x200000" Type="0x62" Instance="0x01">

  <TypeAttrib Copy="0x1" ReadOnly="0x0" RegionType="0x0" ResetImage="0x0"
    Compressed="0x1"/>

</POINT_ENTRY>

```

In previous example, PSP FW copies content from SPI_CS2_L offset 0x750000L to DRAM 0x09A00000 with size 0x200000 and sets reset vector to it. PSP FW also copies content from SPI_CS1_L offset 0x750000L to DRAM 0x09C00000.

- f. Attribute: AddressMode (Optional)—Available only if directory address mode is 2 or 3. This attribute is ignored if the directory address mode is not 2 or 3.

Valid values: 0, 1, 2, 3. Default: the same as directory address mode. Value 3 is valid only in the Entry attribute. It is not valid in the DIRS attribute or the PSP_DIR/BIOS_DIR attribute.

Each AddressMode sets bit63 and bit62 to specific values shown below. Those bits are set only if directory address mode is 2 or 3.

Address Mode 0: Physical Address, bit 63~56: 0x00

Address Mode 1: Relative Address to entire BIOS image, bit 63~56: 0x40

Address Mode 2: Relative Address to PSP/BIOS directory, bit 63~56: 0x80

Address Mode 3: Relative Address to slot N, bit 63~56: 0xC0

6. Node <VALUE_ENTRY>

Specify the entry used as PSP SW fuse

- a. Attribute: Value (Required)
Format in hexadecimal

7. Node <POINT_ENTRY>

Specify an entry with no file required to embed to the reserved region. The tool will fill below information to directory header only.

- a. Attribute: Type (Required)
Specify the type as defined in Table 8, “AMD Family 17h and 19h Processor PSP Directory Type Encodings” on page 44 .

- b. Attribute: Size (Required)
Size to be filled to the Size filed of PSP Directory Table Entry.
- c. Attribute: Address (Required)
Address to be filled to the Location filed of PSP Directory Table Entry
- d. Attribute: RomId (Optional)
Explicit declare which SPI device PSP load from, 0: the SPI device select by SPI_CS1_L, 1: the SPI device select by SPI_CS2_L.
- e. Attribute: AbsoluteAddr (Optional)
This attribute is only valid if the directory has AddressMode = 2.
If AbsoluteAddr = 0x1, while the AddressMode of the directory is 2, then the value of "Address" attribute is considered a physical address (just like AddressMode = 0) instead of a relative address. For example, if the directory base address is 0x140000, and the entry address is 0xFF002000, AbsoluteAddr = 0x1, then the real binary offset is 0x2000.
- f. Attribute: AddressMode (Optional)
Same as image entry

B.1.3 Node <BIOS_DIR>

Specify the entries belong to BIOS Directory. Same as the PSP_DIR, these pair attributes Base and Size are used to reserve a region in the flash to hold the directory header as well as the files to be embed, if these attributes have been defined, the tool will automatically calculate the address for child "IMAGE_ENTRY" node, or else the "Address" should be explicitly declared in every child "IMAGE_ENTRY"

1. Attribute: Base, Headerbase (Optional)

Define the Base of reserved region for BIOS Directory. If "Headerbase" is specified, the directory header structure will place at the region which "Headerbase" point to, or else, the directory header structure and firmware images will place continual which "Base" point to.

With "Headerbase" declared, it allow to separate the PSP Directory header and the region used to hold the firmware.

For example, <BIOS_DIR Level="1" **HeaderBase**="0x146000" **Base**="0x1146000" Size="0x100000" SpiBlockSize="0x1000">

In this example, the PSP directory header structure is placed at address 0x146000, the FW belongs to the PSP Directory is placed at 0x1146000.

This lets the engineer place the PSP directory header in lower 16M of SPI, and PSP FW is in the higher 16M.

2. Attribute: Size (Optional)

Defines the size of the reserved region for BIOS Directory

3. Attribute: Copy (Optional)

This attribute is primarily for A/B recovery use.

If Copy is set to 0x1, the directory will copy all information other than **Base** attribute from the previous Level 2 directory in the same configure file.

If no level 2 directory is found, the tool generates an error.

All attributes of the directory other than **Base** and all nodes are ignored.

The tool picks up the **Base** attribute to set the directory base address.

For example, <PSP_DIR AddressMode="0x2" Base="0x870000" Level="0x2" Size="0x12a000" SpiBlockSize="0x1000" Copy="0x1">.

4. Attribute: AddressMode(Optional)

AddressMode can be set individually for each directory. This overwrites the AddressMode in **DIRS** tag. If AddressMode is not specified, it inherits from the **DIRS** tag.

5. <IMAGE_ENTRY>

Specify the entry need tool patch/embed the file to final image

- a. Attribute: Type (Required)

Specify the type as defined in Table 17. "BIOS Directory Table Entries" on page 57

The "Copy", "ReadOnly", "RegionType", "ResetImage" will be filed as 0 if not explicit define with child TypeAttrib node.

- b. Attribute: File (Required)

Same as in PSP_DIR

- c. Attribute: Size (Optional)
Same as in PSP_DIR
- d. Attribute: Address (Optional)
Same as in PSP_DIR
- e. Attribute: Destination (Optional)
Define the value to be filled to BIOS Directory Entry's Destination field. The default value is 0 if not explicitly defined.
(e.g., Specify APOB need to be filled to destination 0x4000000)
<POINT_ENTRY Type="0x61" Address="0" Size="0" Destination="0x04000000"/>
- f. Attribute: RomId (Optional)
Explicitly declare which SPI device PSP load from, 0: the SPI device selected by SPI_CS1_L, 1: the SPI device selected by SPI_CS2_L.
- g. Attribute: Instance (Optional)
Declare the instance ID of the entry. The value is 0 if not specified.
- h. Attribute: SubProgram (Optional)
Declare the SubProgram ID of the entry. The value is 0 if not specified.
- i. Node <TypeAttrib> (Optional)
Attribute: Copy, ReadOnly, RegionType, ResetImage, Compressed (Required)
Specify the value to be filled to the BIOS Directory Table Entry.
RegionType: Not required for X86 ISA, 0 — Normal memory, 1 TA1 memory, 2 TA2 memory
BIOSResetImage: 1 — BIOS reset binary (PEI volume for x86, BL3 fw for ARM), PSP FW will release X86 at certain address by using other fields of this entry, for details please refer to "Unified Boot Flow"
Copy: 1 — copy BIOS image from source to destination; 0 — Set region attribute based on <ReadOnly, Source, size> attributes
Read Only: Not required for X86 ISA, 1 — Set region to read-only (applicable for ARM-TA1/TA2) — 0: Set region to read/write -->
Compressed: 1 — zlib Compressed image 0 — uncompressed image
(e.g., below example describes a Compressed BIOS image with uncompressed size 0x590000 copy from address 0x30000 to Destination 0x20000000,

Note: For the compressed entry the "size" of entry should be kept as the uncompressed size of the image

```
<!--Compressed PEI/ DXE (uncompressed size 0x590000) copy to 0x20000000 -->
<POINT_ENTRY Address="0x30000" Size="0x590000" Type="0x62"
Destination="0x20000000">
<TypeAttrib Copy="0x1" ReadOnly="0x0" RegionType="0x0" ResetImage="0x0"
Compressed="0x1"/>
</POINT_ENTRY>
```

- j. Attribute: AddressMode (Optional)
Same as in PSP directory

6. <POINT_ENTRY>

Specify an entry with no file required to embed to the reserved region. The tool will fill below information to directory header only.

- a. Attribute: Type (Required)
Same as in BIOS_DIR.IMAGE_ENTRY
- b. Attribute: Size (Required)
Same as in PSP_DIR
- c. Attribute: Address (Required)
Same as in PSP_DIR
- d. Attribute: Destination (Optional)
Same as in BIOS_DIR.IMAGE_ENTRY
- e. Node <TypeAttrib> (Optional)
Same as in BIOS_DIR.IMAGE_ENTRY
(e.g., specify the BIOS Reset image with size 0x200000, and copy to the destination 0x01E00000, the blob will be authenticate due the ResetImage Flag set to 1)

```
<POINT_ENTRY Address="0x600000L" Destination="0x01E00000" Size="0x200000"
Type="0x62">
```

```
<TypeAttrib Copy="0x1" ReadOnly="0x0" RegionType="0x0" ResetImage="0x1"/>
</POINT_ENTRY>
```

- f. Attribute: RomId (Optional)
Explicit declare which SPI device PSP load from, 0: the SPI device select by SPI_CS1_L, 1: the SPI device select by SPI_CS2_L.
- g. Attribute: Instance (Optional)
Same as Instance in BIOS_DIR.IMAGE_Entry
- h. Attribute: SubProgram (Optional)
Same as SubProgram in BIOS_DIR.IMAGE_Entry
- i. Attribute: AddressMode (Optional)
AddressMode can be set individually for each directory. This overwrites the AddressMode in **DIRS** tag. If AddressMode is not specified, it inherits from the **DIRS** tag.
- j. Attribute: AddressMode (Optional)
Same as in PSP directory

B.1.4 Node <COMBO_DIR>

Describe how to declare a combo directory has describe in section 4.1.4.2 PSP Combo Directory BIOS Support.

1. Attribute: Base
Define the Base of reserved region for PSP Directory. It is using the relative address of binary.
2. Attribute: LookUpMode
Define the "look up mode" PSP boot ROM used to select PSP directory, currently only LookUpMode 0 has been validated.

3. < COMBO_ENTRY >

Specify the entry contains the information PSP boot ROM used to locate & load a PSP directory according to installed silicon.

a. Attribute: IdSelect

Define the ID type of attribute "Id", 0 — Compare PSP ID; 1 — Compare chip family ID. Currently, only IdSelect equal to 0 has been validated.

b. Attribute: ID

Define the 32-bit Chip/PSP ID for a specific PSP directory

c. Attribute: Address

Define the address to hold the PSP directory

The example below demonstrates how to declare a Combo directory to support multiple programs.

Each program has its unique ID, and points to an address that contains the PSP directory.

```
<?xml version="1.0" ?>
<DIRS AddressMode="0">
  <COMBO_DIR Base= "0x0003F000" LookUpMode="0">
    <!-- ZP PSP Direcotry -->
    <COMBO_ENTRY IdSelect="0x00" Id="0xBC090000" Address="0x00080000"/>
    <!-- RV PSP Directory -->
    <COMBO_ENTRY IdSelect="0x00" Id="0xBC0A0000" Address="0x00150000"/>
    <!-- RV2 PSP Directory -->
    <COMBO_ENTRY IdSelect="0x00" Id="0xBC0A0100" Address="0x00150000"/>
    <!-- MTS PSP Directory -->
    <COMBO_ENTRY IdSelect="0x00" Id="0xBC0B0500" Address="0xa8000"/>
  </COMBO_DIR>
</DIRS>
```

B.1.5 ISH Header

Specify an ISH header structure at a given location. PSP tool will generate an ISH header according to the info. ISH header is a 32 bytes structure containing 8 attributes. It lies between PSP Level 1 directory and PSP Level 2 directory. PSP Level 1 directory points to an ISH header by entry id 0x48 or 0x4A, then ISH header points to PSP Level 2 directory. There may be several ISH headers in PSP Level 1 directory. An example below:

```
<PSP_DIR AddressMode="0x1" Base="0x21000" Level="0x1" SpiBlockSize="0x10000">
  <POINT_ENTRY Address="0x30000L" Size="0x0" Type="0x48"/>
  <POINT_ENTRY Address="0x31000L" Size="0x0" Type="0x4a"/>
</PSP_DIR>
<ISH_HEADER Base="0x30000L" BootPriority="0xffffffffL" GlitchRetries="0x0L"
  Location="0x80000" PspId="0xbc0d0200L" Reserved_1="0xffffffffL"
  SlotMaxSize="0xffffffffL" UpdateRetries="0xf"/>
<ISH_HEADER Base="0x31000L" BootPriority="0x2" GlitchRetries="0x0L"
  Location="0x1080000" PspId="0xbc0d0200L" Reserved_1="0xffffffffL"
  SlotMaxSize="0xffffffffL" UpdateRetries="0xf"/>
<PSP_DIR AddressMode="0x2" Base="0x80000" ImageBase="0x80000" Level="0x2"
  LevelType="A" Size="0x260000" SpiBlockSize="0x1000">
```

Declare the following attributes to ensure the tool works as expected:

- Attribute: Base
Declare the ISH header location. PSP tool will generate a 32 bytes ISH header at the declared location.
- Attribute: BootPriority
4 bytes
Declare the boot priority of the ISH header.
- Attribute: UpdateRetries:
4 bytes
Declare the update retries of the ISH header.
- Attribute: GlitchRetries:
4 bytes
Declare the glitch retries of the ISH header.
- Attribute: Location:
4 bytes
Declare the location of PSP directory. Note this is different from "Base" attribute. "Location" declares the PSP directory offset the ISH header points to, while "Base" declares the ISH header's own offset.
- Attribute: PspId:
4 bytes
Declare the PSP Id. This is to differ programs in combo BIOS.
- Attribute: SlotMaxSize:
4 bytes
Declare the slot max size.
- Attribute: Reserved_1:
4 bytes
Reserved for future use.

B.2 Command Line Parameters

Usage: BuildPspDirectory.exe [-h] [--version] [-o OUTPUTPATH] {bd,bb,dp}

Tool used to Build PSP DirTable and Embed/Dump PSP entries.

B.2.1 Positional Arguments

{bd,bb,dp}	Type '<subcommand> -h' for help on a specific subcommand
bd	Build Directory table
bb	Build Bios image with PSP entry file embedded
dp	Dump PSP entry of given BIOS image

B.2.2 Optional Arguments

-h, --help	show this help message and exit
--version	show program's version number and exit
-o OUTPUTPATH, --outputpath OUTPUTPATH,	supported by version 2.x and later

B.2.3 Build Directory Table

Build Directory Table

usage: BuildPspDirectory.exe bd [-h] InBiosImage CfgFile

positional arguments:

InBiosImage	Specify the INPUT BIOS image file name
CfgFile	Specify the configure file for build the PSP Directory

optional arguments:

-h, --help	show this help message and exit
------------	---------------------------------

BB -- Build Bios Image with PSP Entry File Embedded

usage: BuildPspDirectory.exe bb [-h] InBiosImage CfgFile OutBiosImage

Positional arguments:

InBiosImage	Specify the INPUT BIOS image file name
CfgFile	Specify the config file for build the PSP Directory
OutBiosImage	Specify the OUTPUT BIOS image file name

Optional arguments:

-h, --help	show this help message and exit
------------	---------------------------------

DP -- Dump PSP Entry of Given BIOS Image

usage: BuildPspDirectory.exe dp [-h] -p PROGRAM [-x] [-b] [-d] InBiosImage

Positional arguments:

InBiosImage Specify the INPUT BIOS image file name

Optional arguments:

-h, --help show this help message and exit
-p PROGRAM, --program PROGRAM Specify the program name, Valid
choices: ZP, RV, SSP
-x, --xml Output the information in XML format to PspDirInfo.xml
-b, --binary Output psp binaries to outputpath
-d, --directory Output PspDirectory.cfg to outputpath

Appendix C Modified Conventional Resume S3 (SMM->SEC->PEI) Design Guideline

C.1 Introduction

Under secure mode, the PSP firmware restores the memory during resume. Hence, the DRAM is already available when the x86 cores come out of reset. The availability of DRAM provides an opportunity to optimize the BIOS resume path. For example, there is no need to perform Cache as RAM initialization. Also, a new CPU MSR is added and any write to this MSR results in CPU context (MSR, microcode, etc.) being automatically saved in protected fenced memory. During resume from sleep transition, the CPU core automatically restores to this CPU context saved in fenced memory; and x86 cores immediately execute from the resume vector that was passed as part of write to MSR during cold boot. The availability of DRAM, and properly restored CPU context at x86 reset time, provide an opportunity to improve BIOS resume time.

The existing UEFI BIOS currently does not comprehend this kind of S3 resume. Conventionally, the BIOS PEI driver is expected to run from ROM code and perform memory restore. Historically, the reset addresses for x86 cold boot and S3-resume were the same (i.e., 0xFFFFFFF0). Hence, during the resume path the BIOS code executed the same PEI driver in SPI space that it would execute during cold boot. However in this case the BIOS resume vector can be the DRAM location and the resume path can be optimized to make use of this fact.

Various possible ways to provide the DRAM resume path are listed below:

- Custom Resume Path, CPU resume from specified resume vector in memory, do all register replay sequence without turn back to SEC, PEI Phase, and handle off to OS Directly.
- Separate Firmware Volume for Resume Code
- SMM Resume, do all the resume sequence include handle off to OS inside SMM. This mode takes the advantage of Security of SMM Memory, Miniature, DXE likes infrastructure
- Modified Conventional Resume (AMD CRB Implementation), Considering the compatible with conventional S3 Resume path and lease the maintain effort, With this approach, CPU will resume back in SMM then jump to SEC and PEI on DRAM to perform conventional S3 resume path, except the PEI stack, and AGESA heap establishes reserved memory region, and some MSR programming step are skipped.

This doc will only focus on the implementation details of Modified Conventional Resume AMD reference board used.

C.2 Design Discussion

With Modified Conventional Resume path, the x86 resumes from sleep, begins executing code from a predefined SMM resume vector and then jump to ROM code to continue conventional resume.

Unique items that need to be addressed within this design include:

- How to set SMM resume vector?
- How to jump from SMM to the address of SEC?

The jump is through execute instruction "RSM" in the resume entry with the SMM save state (GDT, IP, SP and etc) modified.

- How to publish this special resume behavior and heap base address to PEI drivers?

When jump to Sec, EAX is filled with pointer of PSP_SMM_RSM_MEM_INFO structure, then we can publish this structure via install predefine PPI.

EDX is filled with special signature "0x55AABB66" when jump to Sec, this signature can be used to identify if resume back from SMM resume.

SOC 15 FW S3 Flow (including S3 related on SS boot flow)

```

graph TD
    subgraph ABL
        ABL_Init[InitiaDCT_Df, Prepare APOB data]
        ABL_Load[ABL 0 Load ABL 4 if S3 detected]
        ABL_LoadS3[ABL 4 Locate & Load S3 reply buffer from SPI BIOS for entry dualS3]
        ABL_Auth[Authenticate memory contents - Release X BS]
    end

    subgraph PSP
        PSP_Start[System Start]
        PSP_Release[Release X BS]
        PSP_PSP[PSP S3 Process]
        PSP_Load[Load & Release ABL_SMU]
        PSP_Exit[System Exit S3]
    end

    subgraph SMU
        SMU_Enter[System Enter S3]
    end

    subgraph BIOS
        BIOS_PEI[BIOS PEI Phase]
        BIOS_DUE[BIOS DUE Phase]
        BIOS_BDS[BIOS BDS Phase]
        BIOS_Boot[OS Boot]
        BIOS_Execute[OS execute _JTAG_ACH method]
        BIOS_Restore[Locate restore core contents & jump to resume vector of MSRC 001_03D0]
        BIOS_RSP[RSP start from B platform resume vector]
        BIOS_RSP_Check[RSP check if resume from secure S3]
        BIOS_PEI_Phase[PEI Phase Execute from resume DRAM]
        BIOS_DUE_IPL[DUE IPL Execute from resume DRAM]
        BIOS_Resumed[OS resumed]
    end

    ABL_Init --> PSP_Release
    PSP_Release --> PSP_PSP
    PSP_PSP --> BIOS_PEI
    BIOS_PEI --> BIOS_DUE
    BIOS_DUE --> BIOS_BDS
    BIOS_BDS --> BIOS_Boot
    BIOS_Boot --> BIOS_Execute
    BIOS_Execute --> SMU_Enter
    SMU_Enter --> PSP_Exit
    PSP_Exit --> ABL_Load
    ABL_Load --> ABL_LoadS3
    ABL_LoadS3 --> ABL_Auth
    ABL_Auth --> BIOS_PEI
    BIOS_PEI --> BIOS_Restore
    BIOS_Restore --> BIOS_RSP
    BIOS_RSP --> BIOS_RSP_Check
    BIOS_RSP_Check --> BIOS_PEI_Phase
    BIOS_PEI_Phase --> BIOS_DUE_IPL
    BIOS_DUE_IPL --> BIOS_Resumed
  
```

ABL

- InitiaDCT_Df, Prepare APOB data
- ABL 0 Load ABL 4 if S3 detected
- ABL 4 Locate & Load S3 reply buffer from SPI BIOS for entry dualS3
- Authenticate memory contents - Release X BS

PSP

- System Start
- Load & Release ABL & SMU
- Release X BS
- PSP S3 Process
- Load & Release ABL_SMU
- System Exit S3

SMU

- System Enter S3

BIOS

- BIOS PEI Phase**
 - Platform PEI driver execution
 - Request PEI DRAM region reserved
 - AGESA V3 PEI driver execution
 - Release APOB region after S3 exit boot service
- BIOS DUE Phase**
 - Platform DUE driver execution
 - Secure S3 not required to S3TFS - 3 script
 - Request 3 resume IPan DRAM & stack size
 - through mhp/hp/hp/ndm/hp/ndm of AGESA V6 DUE Driver execution
 - Secure to S3TFS 3 script through APOB - Release X
 - PSP driver save APOB to SPI ABL for entry dualS3
 - PSP driver prepare S3 exit SMM environment (Platform.c)
- BIOS BDS Phase**
 - Ready to boot event
 - MemoryMISC 001_03D0, Ucode will save core snapshot for S3 Exit
 - *Send S3Save to save the Post SMM/loc 5_25 script operation through SMM communication service
- OS Boot**
- OS execute _JTAG_ACH method
- OS write Sleep Type register
- BIOS execute Sleep Type trap SM Handler
- Send Message to PSP
- Disable Sleep Type SM trap
- Send SM U Message

Appendix C Modified Conventional Resume S3 (SMM->SEC->PEI) Design Guideline

C.3 Platform BIOS Porting Details

Below items are needed to implement support of Modified Conventional S3:

The DXE_SMM_DRIVER instance of AmdPspFlashAccLib is implemented by platform BIOS, and this library is aid to provide generic interface for AGESA module to access SPI flash including Read/Write/Erase/GetBlockSize.

- **AMD_PSP_PLATFORM_PROTOCOL_GUID (PspPlatformDriver.c)**

This protocol allows platform BIOS to customize the stack and Heap size for PEI that we will build Stack and Heap on memory instead CAR.

And the most important field of this protocol is the GdtOffset, will patch to SMM Save area before calling "RSM" to achieve jump form SMM to SEC. The Gdt table can be found via put in fix address Or by search specific signature. In CRB design, we will put the GDT table above the "Flat32Start"

See below Security code section for details

- **Update Security codes**

Note: Alternate way can be used to achieve below requirement, The following sample codes are just provide as reference:

Note: *Clear Long Mode Enable*

if SMM is @64bit mode while PEI is under 32bit mode, we need to clear this bit to avoid unexpected behavior

Sample code:

```
;Clear Long Mode Enable
mov ecx, 0c0000080h ; EFER MSR number.
rdmsr                ; Read EFER.
btr eax, 8           ; Set LME=0
wrmsr                ; Write EFER.
```

Put GDT Table

```
;; Offset 0xFFFFFFFF0 (reset address has code below>>
;; nop; nop; jmp Flat32Start
;; So we can read offset 0xFFFFFFFF3 to find the location of Flat32Start
;; Subtract that with 12 we will know this SMMResumeInfo location
SMMResumeInfo: ;; This offset can bie found as 0xFFFFFFFF5 + word [ 0xFFFFFFFF3] -
12
DD OFFSET BootGDTtable          ; GDT base address
DW LINEAR_CODE_SEL              ; code segment
DW SYS_DATA_SEL                ; data segment
DD OFFSET ProtectedModeEntryPoint ; Offset of our 32 bit code
DD SMM_RESUME_SIGNATURE
Flat32Start PROC NEAR C PUBLIC
```

Identify SMM Resume Path

EDX will be set to 0x55AABB66, when exit from PSP resume entry.

Sample code:

```

SMM_RESUME_SIGNATURE = 055AABB66h
cmp     edx, SMM_RESUME_SIGNATURE
je      SmmS3Resume
jmp     xxx
SmmS3Resume:

```

Publish PSP_SMM_RSM_MEM_INFO @BSP Through PPI

EAX will be set to the pointer of PSP_SMM_RSM_MEM_INFO structure when exit from PSP resume entry

Sample code:

```

        push     EAX                                ;Address of mPspSmmRsmMemInfo
        mov     edi, PEI_CORE_ENTRY
        push    DWORD PTR ds:[edi]
        mov     edi, BFV_BASE_ADDRESS
        push    DWORD PTR ds:[edi]
        push    BSP_STACK_SIZE_64K
        call    SecStartupResume

EFI_PEI_PPI_DESCRIPTOR mPspSmmResumePpi =
{
    (EFI_PEI_PPI_DESCRIPTOR_PPI | EFI_PEI_PPI_DESCRIPTOR_TERMINATE_LIST),
    &gPspSmmResumePpiGuid,
    NULL
};

VOID
SecStartupResume (
    UINT32                SizeOfRam,
    UINT32                BootFirmwareVolume,
    PEI_MAIN_ENTRY_POINT  PeiCoreEntryPoint,
    PSP_SMM_RSM_MEM_INFO *PspSmmRsmMemInfo
)
{
    EFI_PEI_STARTUP_DESCRIPTOR  PeiStartup;
    PSP_SMM_RESUME_PPI *PspSmmResumePpi;
    EFI_PEI_PPI_DESCRIPTOR *PspSmmResumePpiDesc;
    //Init PspSmmResumePpiDesc
    PspSmmResumePpiDesc = (EFI_PEI_PPI_DESCRIPTOR *) (UINTN) PspSmmRsmMemInfo->
TempRegionPtr;
    EfiCommonLibCopyMem (PspSmmResumePpiDesc, &mPspSmmResumePpi, sizeof
(mPspSmmResumePpi));
    PspSmmResumePpi = (PSP_SMM_RESUME_PPI *) (PspSmmResumePpiDesc + 1);
    PspSmmResumePpi->HeapBase = PspSmmRsmMemInfo->RsmHeapPtr;
    PspSmmResumePpiDesc->Ppi = PspSmmResumePpi;
}

```

```

PeiStartup.SizeOfCacheAsRam    = SizeOfRam;
PeiStartup.BootFirmwareVolume = BootFirmwareVolume;
PeiStartup.DispatchTable      = PspSmmResumePpiDesc;

//
// Transfer the control to the PEI core
//
(*PeiCoreEntryPoint) (&PeiStartup);
return ;
}

```

Force AP to halt in Security phase:

If the platform BIOS requires a combo support, family check is required due the difference of S3 implementation of pre-SOC15 and SOC15.

Sample codes:

```

StartUpApS3      PROC      NEAR      PUBLIC
    mov          eax, 1
    cpuid
    shr          eax, 20
    and          eax, 000000FFh
    cmp          eax, 06h          ; Family 15h?
    je           SkipApHlt
@@:
    cli          ; Family 17h AP just halt here
    hlt
    jmp @b

```


Appendix D Postcode Definition for PSP FW

PSP FW postcode definition files are released with AGESA PI package, PSP bootloader postcode definition can be founded at path: AgesaModulePkg\Firmwares\<Program>\bl_errorcodes.h. PSP AGESA bootloader postcode definition can be founded at path: AgesaModulePkg\Firmwares\<Program>\AblPostCode.h.

Appendix E HSTI Bitmap Definition

HSTI is short for Hardware Security Testability Specification. It is a specification defined by Microsoft®, used to protect against misconfiguration of security features on Windows® devices.

Per specification required, as an IHV, AMD defines a bit field representing the testable security features of the platform, details as below.

E.1 Security Feature Byte Index 0

Bit 0	Do you use RSA 2048 and SHA256 only (or similar crypto strength)
-------	--

E.2 Security Feature Byte Index 1

Firmware Code must be present in protected storage.

Bit 0	Do you protect spiflash?
Bit 1	Do you implement read-only until reset for eMMC FW partitions? Please fill the implemented, verified bits as 1, if no eMMC device attached
Bit 2	Do you support Signed Firmware, Check Firmware that is installed by OEM is either read only or protected by secure firmware update process

E.3 Security Feature Byte Index 2

Secure firmware update process

Bit 0	Is secure firmware update process on by default with test keys? (RECOMMENDED)
Bit 1	Do you check whether test keys are used in production?
Bit 2	Do you allow rollback to insecure firmware version? If yes then what is the protection mechanism? Do you clear TPM when rollback happens?

E.4 Security Feature Byte Index 3

Do you have backdoors to override SecureBoot

Bit 0	Does your system support inline prompting to bypass Secureboot? If yes then is it disabled while SecureBoot is enabled
Bit 1	Do you have manufacturing backdoors? Do you check for them to be disabled while SecureBoot is enabled and always disabled in production system?

E.5 Security Feature Byte Index 4

AMD specific security feature

Bit 0	Fuse SecureEnable to ensure silicon level secure
Bit 1	PSP platform secure boot enable, support boot Integrity
Bit 2	PSP debug lock enable, Protection against external hardware debugger

E.6 Security Feature Byte Index 5

Compatibility Support Modules (CSM)

Bit 0	Do you check blocking of loading CSM when SecureBoot is enabled
Bit 1	Do you check if CSM is not present on CS systems permanently? Please fill the implemented, verified bits as 1 for non CS system.

Appendix F FIPS Certification on AMD FP6 Platform

The Federal Information Processing Standards (FIPS) are a set of US Government security requirements for data and its encryption.

This appendix is used as a guideline for an IBV/OEM to make a FIPS-capable BIOS on the AMD FP6 platform.

Supported CPU families: This appendix is applicable only to the mobile FP6 platform with AMD Family 17h Models 60h-67h processors. It does not apply to FIPS for desktop.

Enable FIPS mode:

1. Reserve at least 16KB for PSP level 1 directory because FIPS requires an additional 8KB ROM space for the PSP L1 Bootloader.
2. Platform BIOS must include the file with “_FIPS” at the end of the filename as PSP entry 0x1. For example:

The file `PspBootLoader_stagel_prod_AB_RN_FIPS.sbin` is a FIPS-capable PSP boot loader.
The file `PspBootLoader_stagel_prod_AB_RN.sbin` is not a FIPS-capable PSP boot loader.

3. Set BIT 32 of the PSP soft fuse chain (PSP entry 0xB) to enable FIPS certification.

Definition of BIT32 in PSP entry 0xB:

FIPS certification enablement:

0: FIPS certification mode is OFF

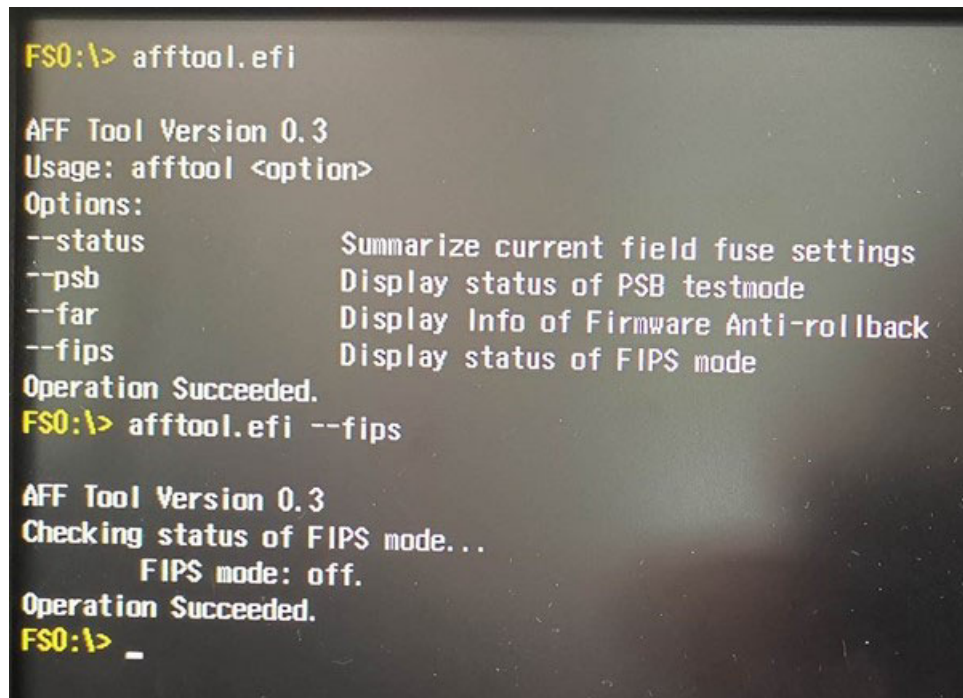
1: FIPS certification mode is ON

Verify FIPS mode is on:

1. Boot the system into a UEFI shell with secure boot disabled. Use the shell version of Afftool version 0.3 or later.
2. Run the afftool with the command: `afftool -fips`

If it shows `FIPS mode: on`, this is a FIPS-capable BIOS.

This image in Figure 14 shows that FIPS mode is disabled.



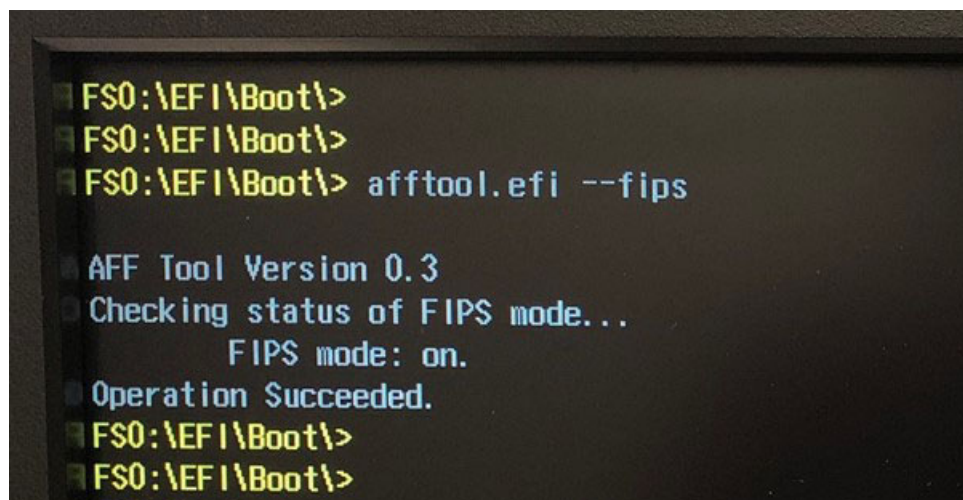
```
FS0:\> afftool.efi

AFF Tool Version 0.3
Usage: afftool <option>
Options:
--status          Summarize current field fuse settings
--psb             Display status of PSB testmode
--far            Display Info of Firmware Anti-rollback
--fips           Display status of FIPS mode
Operation Succeeded.
FS0:\> afftool.efi --fips

AFF Tool Version 0.3
Checking status of FIPS mode...
      FIPS mode: off.
Operation Succeeded.
FS0:\> _
```

Figure 14. FIPS Mode Disabled

This image in Figure 15 shows that FIPS mode is enabled and passed self-tests.



```
FS0:\EFI\Boot>
FS0:\EFI\Boot>
FS0:\EFI\Boot> afftool.efi --fips

AFF Tool Version 0.3
Checking status of FIPS mode...
      FIPS mode: on.
Operation Succeeded.
FS0:\EFI\Boot>
FS0:\EFI\Boot>
```

Figure 15. FIPS Mode Enabled, Self-Tests Passed