



Kaseyville Intel® Firmware Support Package (FSP) Integration Guide

Sat May 2 2026 21:56:18

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Any software source code reprinted in this document is furnished for informational purposes only and may only be used or copied and no license, express or implied, by estoppel or otherwise, to any of the reprinted source code is granted by this document.

[When the doc contains software source code for a special or limited purpose (such as informational purposes only), use the conditionalized Software Disclaimer tag. Otherwise, use the generic software source code disclaimer from the Legal page and include a copy of the software license or a hyperlink to its permanent location.]

This document contains information on products in the design phase of development. Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. ("products") in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as "commercial" names for products. Also, they are not intended to function as trademarks.

Intel, Intel Atom, [include any Intel trademarks which are used in this document] and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright ©Intel Corporation. All rights reserved.

Contents

1	Introduction	1
1.1	Purpose	1
1.2	Intended Audience	1
1.3	Related Documents	1
1.4	Acronyms and Terminology	1
2	Overview	3
2.1	Technical Overview	3
2.2	FSP Distribution Package	3
2.2.1	Package Layout	4
3	FSP Integration	5
3.1	Assumptions Used in this Document	5
3.2	Boot Flow	5
3.3	FSP INFO Header	5
3.4	FSP Image ID and Revision	5
3.5	FSP Global Data	6
3.6	Memory Map	6
4	FSP Dispatch Mode	7
4.1	Overview	7
4.2	Dispatch Mode Policy Init	8
4.2.1	FSP-M Policy Initialization Flow	8
4.2.2	FSP-S Policy Initialization Flow	9
4.3	Config Blocks	9
4.3.1	Overview	9
4.3.2	Config Block Data Format	9
4.3.3	Config Block Library APIs	11
4.3.4	Config Blocks used by FSP-M	11
4.3.5	Config Blocks used by FSP-S	11
4.4	FSP Error Information	12
4.5	Dispatch Mode Integration	13
5	FSP API Mode	15
5.1	Overview	15
5.2	FSP APIs	15
5.2.1	TempRamInit API	15
5.2.2	FspMemoryInit API	15
5.2.3	TempRamExit API	16
5.2.4	FspSiliconInit API	16
5.2.5	NotifyPhase API	17
5.2.5.1	PostPciEnumeration Notification	17
5.2.5.2	ReadyToBoot Notification	17
5.2.5.3	EndOfFirmware Notification	17
5.3	Reset Return Codes	18
5.4	UPD Porting Guide	18

6	Porting Recommendations	19
6.1	Locking PAM register	19
6.2	Locking SMRAM register	19
6.3	Locking SMI register	19
6.4	Verify below settings are correct for your platforms	20
7	FSP Output	21
7.1	FSP_ERROR_INFO_HOB	21
7.2	CXL_CEDT_HOB	22
7.3	CXL_NODE_SOCKET	22
7.4	IIO_UDS	22
7.5	SYSTEM_MEMORY_MAP_HOB	22
7.6	PREV_BOOT_ERR_SRC_HOB	23
7.7	SYSTEM_INFO_VAR	23
7.8	FSP_EXT_MEMORY_PPR_HOB	23
7.9	FSP_EXT_SYSTEM_MEMORY_MAP_HOB	23
8	POST Codes	25
8.1	POST Code Info	25
8.1.1	TempRamInit API Status Codes (0xFxxx)	26
8.1.2	FspMemoryInit API Status Codes (0xDxxx)	26
8.1.3	TempRamExit API Status Codes (0xBxxx)	26
8.1.4	FspSiliconInit API Status Codes (0x9xxx)	26
8.1.5	NotifyPhase API Status Codes (0x6xxx)	26
9	Class Index	27
9.1	Class List	27
10	File Index	29
10.1	File List	29
11	Class Documentation	31
11.1	CEDT_CFMWS_WINDOW_RESTRICTIONS Union Reference	31
11.1.1	Detailed Description	31
11.2	CXL_CEDT_HOB Struct Reference	31
11.2.1	Detailed Description	31
11.3	FSP_ERROR_INFO_HOB Struct Reference	31
11.3.1	Detailed Description	32
11.4	PLATFORM_DATA Struct Reference	32
11.4.1	Detailed Description	33
11.5	UDS_PCIROOT_RES Struct Reference	33
11.5.1	Detailed Description	33
11.6	UDS_SOCKET_RES Struct Reference	34
11.6.1	Detailed Description	34
11.7	UDS_STACK_RES Struct Reference	34
11.7.1	Detailed Description	35
12	File Documentation	37
12.1	CxlCedtHob.h File Reference	37
12.1.1	Detailed Description	37
12.2	CxlNodeHob.h File Reference	37
12.2.1	Detailed Description	38
12.3	FspAcpiHobs.h File Reference	38
12.3.1	Detailed Description	38
12.4	FspEdpcParam.h File Reference	38
12.4.1	Detailed Description	38
12.5	FspErrorInfoHob.h File Reference	39
12.5.1	Detailed Description	39
12.6	FspExtMemoryPprHob.h File Reference	40

12.6.1 Detailed Description	40
12.7 FspExtSystemMemoryMapHob.h File Reference	40
12.7.1 Detailed Description	41
12.8 FspGlobals.h File Reference	41
12.8.1 Detailed Description	42
12.9 FspiUpd.h File Reference	42
12.9.1 Detailed Description	43
12.10FspmUpd.h File Reference	43
12.10.1 Detailed Description	44
12.11FspUpd.h File Reference	44
12.11.1 Detailed Description	44
12.12FspUpd.h File Reference	44
12.12.1 Detailed Description	45
12.13FspUpd.h File Reference	45
12.13.1 Detailed Description	46
12.14lioPcieConfigUpd.h File Reference	46
12.14.1 Detailed Description	46
12.15lioUniversalDataHob.h File Reference	47
12.15.1 Detailed Description	47
12.16MemoryMapDataHob.h File Reference	47
12.16.1 Detailed Description	47
12.16.2 Macro Definition Documentation	48
12.16.2.1 MEM_SKTCH_TO_IMC	48
12.17PrevBootErrSrcHob.h File Reference	48
12.17.1 Detailed Description	48
12.18SystemInfoHob.h File Reference	49
12.18.1 Detailed Description	49
Index	51

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to describe the steps required to integrate the Intel® Firmware Support Package (FSP) into a boot loader solution. It supports **Kaseyville** platforms with **GraniteRapids** processor.

1.2 Intended Audience

This document is targeted at all platform and system developers who need to consume FSP binaries in their boot loader solutions. This includes, but is not limited to: system BIOS developers, boot loader developers, system integrators, as well as end users.

1.3 Related Documents

- *Platform Initialization (PI) Specification v1.7* located at <http://www.uefi.org/specifications>
- *Intel® Firmware Support Package: External Architecture Specification (EAS) v2.3* located at <https://cdrdv2.intel.com/v1/dl/getContent/644852>.
- *Boot Setting File Specification (BSF) v1.0* https://firmware.intel.com/sites/default/files/BSF_1_0.pdf
- *Binary Configuration Tool for Intel® Firmware Support Package* available at <http://www.intel.com/fsp>

1.4 Acronyms and Terminology

Acronym	Definition
BCT	Binary Configuration Tool
BDSM	Base Data Of Stolen Memory
BSF	Boot Setting File
BSP	Boot Strap Processor
BWG	BIOS Writer's Guide
CAR	Cache As Ram
CRB	Customer Reference Board
DPR	DMA Protected Range

Acronym	Definition
FIT	Firmware Interface Table
FSP	Firmware Support Package
FSP API	Firmware Support Package Interface
FW	Firmware
GTT	Graphics Translation Table
IFWI	Integrated Firmware Image
IOT	Internal Observation Trace
MRC	Memory Reference Code (Memory Init code encapsulated by FSP-M)
MOT	Memory Observation Trace
PCH	Platform Controller Hub
PMC	Power Management Controller
PMRR	Protected Memory Range Reporting
REMAP	Remapped Memory Area
RVP	Reference and Validation Platform
SBSP	System BSP
SMI	System Management Interrupt
SMM	System Management Mode
SMRAM	System Management Mode RAM
SPI	Serial Peripheral Interface
TOLUD	Top of Low Usable Memory
TOUUD	Top of Upper Usable Memory
TSEG	Memory Reserved at the Top of Memory to be used as SMRAM
UPD	Updatable Product Data

Chapter 2

Overview

2.1 Technical Overview

The *Intel® Firmware Support Package (FSP)* provides chipset and processor initialization in a format that can easily be incorporated into many existing bootloaders.

The FSP will perform the necessary initialization steps as documented in the BWG including initialization of the CPU, memory controller, chipset and certain bus interfaces, if necessary.

FSP is not a stand-alone boot loader; therefore it needs to be integrated into a host boot loader to carry out other boot loader functions, such as: initializing non-Intel components, conducting bus enumeration, and discovering devices in the system and all industry standard initialization.

The FSP binary can be integrated easily into many different boot loaders, such as Coreboot, EDKII MinPlatform, Intel® Slim Bootloader, etc. and also into the embedded OS directly.

Below are some required steps for the integration:

- **Customizing** The static FSP configuration parameters are part of the FSP binary and can be customized by tools provided by Intel. This step is optional as configuration data may also be provided at runtime.
- **Rebasing** The FSP is not Position Independent Code (PIC) and the whole FSP has to be rebased if it is placed at a location which is different from the preferred address during build process.
- **Placing** Once the FSP binary is ready for integration, the boot loader build process needs to be modified to place this FSP binary at the specific rebasing location identified above.
- **Interfacing** The boot loader needs to add code to setup the operating environment for the FSP, call the FSP with correct parameters and parse the FSP output to retrieve the necessary information returned by the FSP.

2.2 FSP Distribution Package

- The FSP distribution package contains the following:
 - FSP Binary
 - FSP Integration Guide
 - BSF Configuration File (BSF)
 - Data Structure Header Files
- The FSP configuration utility called BCT is available as a separate package. It can be downloaded from link mentioned in Section 1.3.

2.2.1 Package Layout

- **Docs (Auto generated)**

- Kaseyville_FSP_Integration_Guide.pdf
- Kaseyville_FSP_Integration_Guide.chm

- **Include**

- [FspUpd.h](#), [FspmUpd.h](#), [FspUpd.h](#) and [FspiUpd.h](#) (FSP UPD structure and related definitions)
- [FspUpd.h](#)
- [CxlCedtHob.h](#)
- [CxlNodeHob.h](#)
- [FspErrorInfoHob.h](#)
- [FspExtMemoryPprHob.h](#)
- [FspExtSystemMemoryMapHob.h](#)
- [FspGlobals.h](#)
- [IioPcieConfigUpd.h](#)
- [IioUniversalDataHob.h](#)
- [MemoryMapDataHob.h](#)
- [PrevBootErrSrcHob.h](#)
- [SystemInfoHob.h](#)

- BirchStreamFspBinPkg.dec (EDKII declaration file for package)

- Fsp.fd (FSP Binary)

- Server.yaml
-

Chapter 3

FSP Integration

3.1 Assumptions Used in this Document

The FSP for this platform is built with a preferred base address given by `PcdFspAreaBaseAddress` and so the reference code provided in the document assumes that the FSP is placed at this base address during the final boot loader build. Users may rebase the FSP binary at a different location with Intel's Binary Configuration Tool (BCT) or `SplitFspBin.py` before integrating to the boot loader.

For other assumptions and conventions, please refer to Chapter 9 and 10 of the FSP External Architecture Specification version 2.4.

3.2 Boot Flow

Please refer Chapter 7 in the FSP External Architecture Specification version 2.4 for Boot flow chart. The FSP for this platform supports dispatch mode, see Chapter 7 and 10 of the FSP External Architecture Specification version 2.4 for a description of dispatch mode.

3.3 FSP INFO Header

The FSP has an Information Header that provides critical information that is required by the bootloader to successfully interface with the FSP. The structure of the FSP Information Header is documented in the FSP External Architecture Specification version 2.4 with a HeaderRevision of 7.

3.4 FSP Image ID and Revision

FSP information header contains an Image ID field and an Image Revision field that provide the identification and revision information of the FSP binary. It is important to verify these fields while integrating the FSP as API parameters could change over different FSP IDs and revisions. All the FSP FV segments (FSP-T, FSP-M and FSP-P-S) must have same FSP Image ID and revision number, using FV segments with different revision numbers in a single FSP image is not valid. The FSP API parameters documented in this integration guide are applicable for the Image ID and Revision specified as below.

The FSP `ImageId` string in the FSP information header is given by **`PcdFspImageIdString`** and the `ImageRevision` field is given by `SiliconInitVersionMajor|Minor|FspVersionRevision|FspVersionBuild` (Ex:0x0A001460).

3.5 FSP Global Data

FSP uses some amount of TempRam area to store FSP global data which contains some critical data like pointers to FSP information headers and UPD configuration regions, FSP/Bootloader stack pointers required for stack switching etc. HPET Timer register(2) PcdGlobalDataPointerAddress is reserved to store address of this global data, and hence boot loader should not use this register for any other purpose. If TempRAM initialization is done by boot loader, then HPET has to be initialized to the base so that access to the register will work fine.

3.6 Memory Map

Below diagram represents the memory map allocated by FSP including the FSP specific regions.

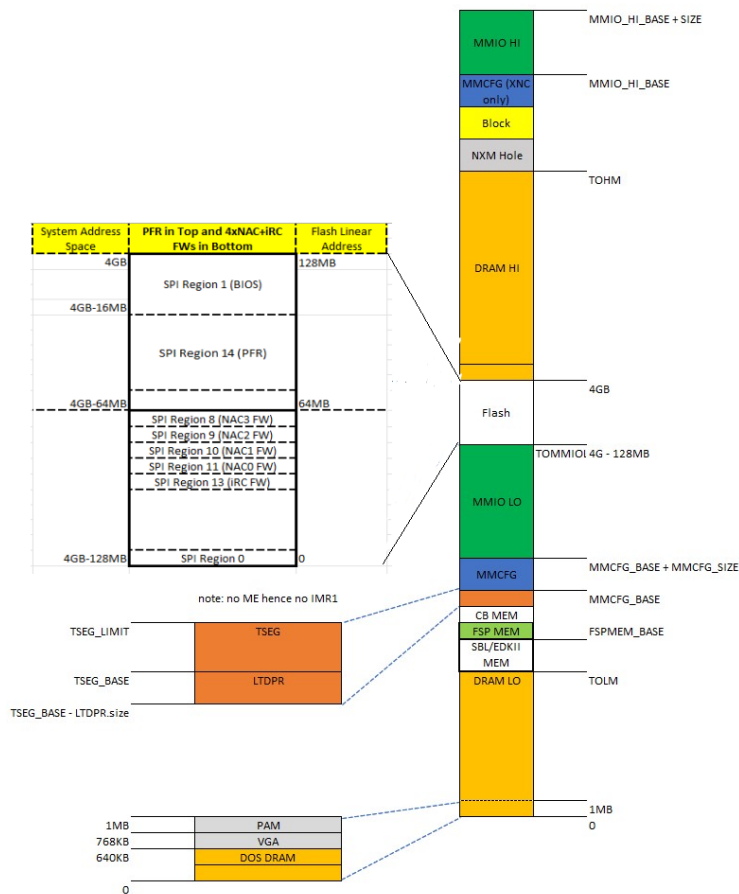


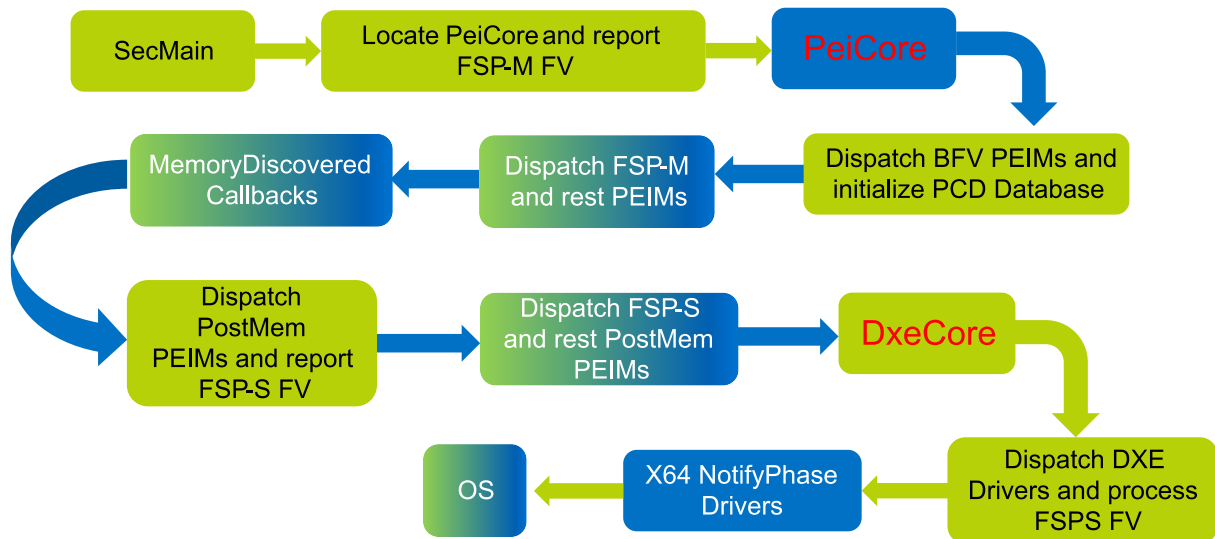
Figure 3.1: System Memory Map

Chapter 4

FSP Dispatch Mode

4.1 Overview

The FSP for this platform supports dispatch mode. Support for dispatch mode can be detected by checking if `FSP_INFO_HEADER->ImageAttribute[BIT1] == 1`. Dispatch mode is intended to enable FSP to integrate well in to UEFI bootloader implementations. Dispatch mode implements a boot flow that is as close to a standard UEFI boot flow as possible. In dispatch mode, the FSP is exposed as standard Firmware Volumes (FVs) directly to the bootloader. The PEIMs in these FVs are executed in the same PEI environment as the boot loader. In dispatch mode, the PPI database, PCD database, and HOB list are shared between the boot loader and the FSP.



Blue blocks are from the FSP binary and green blocks are from the bootloader. Blocks with mixed colors indicate that both bootloader and FSP modules are dispatched during that phase of the boot flow. In dispatch mode, the `NotifyPhase()` API is not used. Instead, FSP-S contains DXE drivers that implement the native callbacks on equivalent events for each of the `NotifyPhase()` invocations.

In dispatch mode, the the PPI database and PCD database are used for providing policy data from the bootloader to FSP. Because these mechanisms provide a great deal of flexibility, dispatch mode does not constrain the method for passing policy data as strongly as API mode. The following sections describe the dispatch mode policy initialization flow used specifically for this platform.

4.2 Dispatch Mode Policy Init

For the plurality of platform designs, most of the policy options provided by FSP do not need to be modified by the bootloader.

To ease this common case, policy initialization has been broken in to two phases: 1. Policy Init and 2. Policy Update.

Policy Init creates the policy data structures with all default policy values pre-populated. Policy Init is implemented using the **factory method pattern**. The FSP provides an API to the bootloader for constructing the policy data structures. Because the factory method is provided by the FSP, backwards compatibility is made substantially easier. The FSP can be assured that the policy data structures match the definitions used at the time the FSP was compiled. As long as new fields are added to the bottom of policy structures, the bootloader may continue to use an older version of the policy data structures at compile time and retain compatibility with both new and old FSP binaries.

There are two factory methods, one for FSP-M and one for FSP-S. `SiliconPolicyInitPreMem()` is provided by FSP-M, `SiliconPolicyInitPostMem()` is provided by FSP-S. These methods are exposed to the bootloader using PPIs. The usage of PPIs ensures API stability between the FSP and the bootloader.

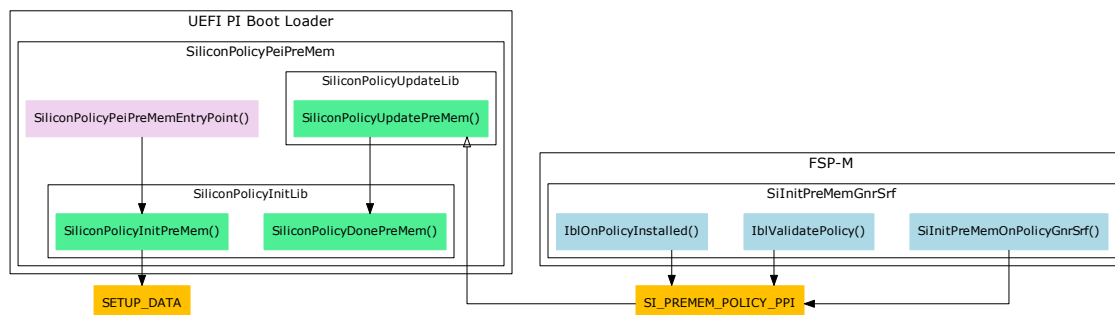
While Policy Init is implemented by the FSP, Policy Update is implemented by the bootloader. Policy Update uses a read-modify-write operation to apply any motherboard specific settings to the policy data while leaving the default values intact when appropriate. After Policy Update is done, FSP may run its SOC initialization code.

4.2.1 FSP-M Policy Initialization Flow

The follow graph shows the policy initialization flow for FSP-M.

Note

The hollow arrow heads in the flow diagram below indicate that action by the PEI Foundation is required for the flow to proceed.



Execution of FSP-M begins after `FspmWrapperInit()` in `IntelFsp2WrapperPkg/FspmWrapperPeim/FspmWrapperPeim.c` installs a `EFI_PEI_FIRMWARE_VOLUME_INFO_PPI` instance for FSP-M. This causes the PEI foundation to become aware of the FV(s) in FSP-M, which schedules all PEIMs in FSP-M for dispatch by PEI. This results in the `SilInitPreMemGnrSrf` PEIM in FSP-M being executed.

One of the actions performed by the `SilInitPrePolicyGnrSrf` entry-point is calling the `SilInitPreMemOnPolicyGnrSrf()` function, which locates the `SI_PREMEM_POLICY_PPI`. The function also, validates and initialize IBL policies after `SI_PREMEM_POLICY_PPI` is located.

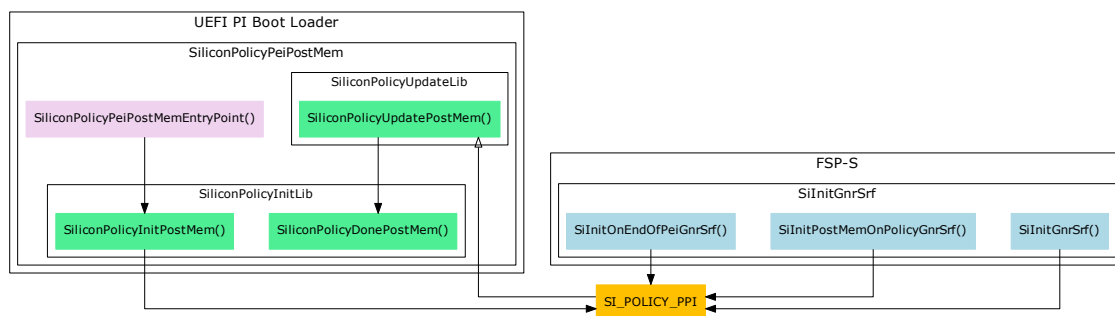
Next `SiliconPolicyPeiPreMem` PEIM will run. `SiliconPolicyPeiPreMemEntryPoint()` in `MinPlatformPkg/PlatformInit/SiliconPolicyPei/SiliconPolicyPeiPreMem.c` will call `SiliconPolicyInitPreMem()` in `BirchStreamOpenBoardPkg/Library/SiliconPolicyInitLib/SiliconPolicyInitLib.c`. This function will get and allocate memory for `SetupDataPtr`, and build a HOB to cache the setup data. With the help of `SetupDataPtr`, all the Intel PEI Platform Policies will be initialized and return it to `SiliconPolicyPeiPreMemEntryPoint()`.

Next, `SiliconPolicyPeiPreMemEntryPoint()` takes the pointer to the policy data returned by `SiliconPolicyInitPreMem()` and passes it to `SiliconPolicyUpdatePreMem()`. `SiliconPolicyUpdatePreMem()` is part of `SiliconPolicyUpdateLib`, which is statically linked to `SiliconPolicyPeiPreMem`. `SiliconPolicyUpdateLib` is special since it is only piece of motherboard specific code in this flow. An example of `SiliconPolicyUpdatePreMem()` is seen in `BirchStreamOpenBoardPkg/Library/SiliconPolicyUpdateLib/SiliconPolicyUpdateLib.c`. `SiliconPolicyUpdatePreMem()` uses a read-modify-write operation to apply any motherboard specific settings to the policy data while leaving the default values intact when appropriate.

After `SiliconPolicyUpdatePreMem()` applies any motherboard specific updates, `SiliconPolicyPeiPreMemEntryPoint()` calls `SiliconPolicyDonePreMem()` in `BirchStreamOpenBoardPkg/Library/SiliconPolicyInitLib/SiliconPolicyInitLib.c`. `SiliconPolicyDonePreMem()` finalizes the silicon pre-mem policy. Silicon code can do initialization based upon the policy data.

4.2.2 FSP-S Policy Initialization Flow

Policy Initialization for FSP-S follows essentially the same flow as FSP-M. The primary difference is function and PPI names do not include the "PreMem" qualifier.



4.3 Config Blocks

4.3.1 Overview

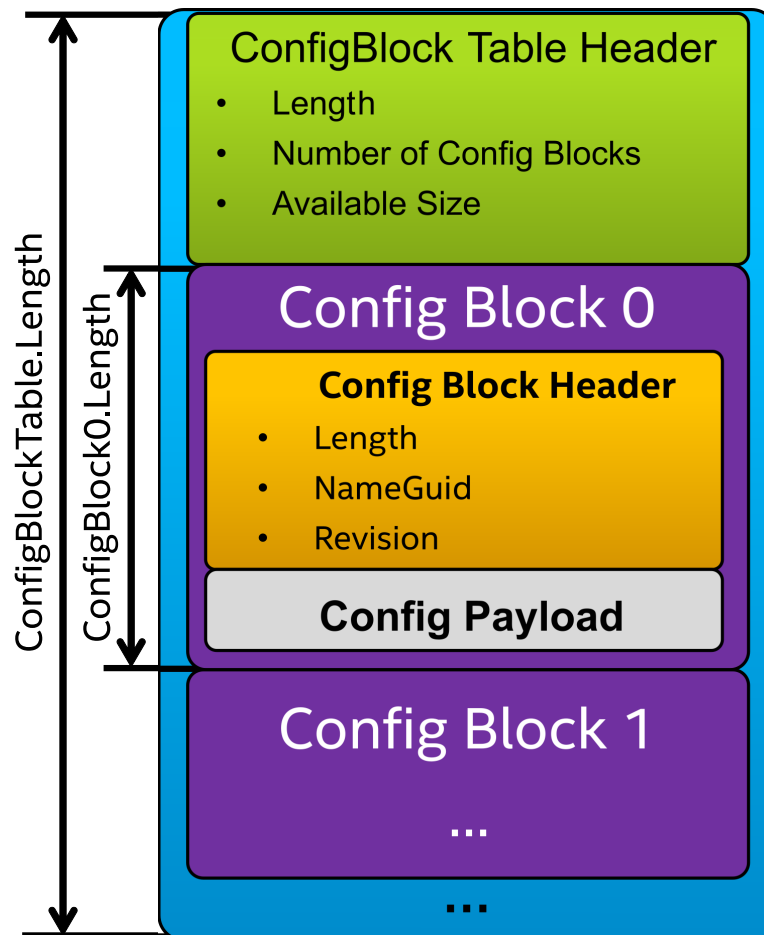
Config Blocks are a binary data serialization format with several unique properties that are useful for firmware implementations.

The serialization format is simple to implement in the C language, minimizing the code needed to implement it; this minimizes the size of the FSP binary. Second, the format is binary backwards compatible; enabling the bootloader to use an older version of the data structure at compile time and retain compatibility with both new and old FSP binaries. Third, the format is mutable in memory without requiring the data to be re-serialized. Finally, the format is position independent with no need for a base address or a rebase operation. This allows the data structure to be moved from pre-memory to post-memory without applying any fixups.

Data in `SI_PREMEM_POLICY_PPI` and `SI_POLICY_PPI` are stored using the Config Block format.

4.3.2 Config Block Data Format

In essence, Config Blocks are multiple C structures that are stored in a single continuous memory range using run length encoding.



The overall structure is termed a Config Block Table. The Config Block Table starts with a `CONFIG_BLOCK_TABLE_HEADER` structure. This header indicates the number of Config Blocks in the Config Block Table, the total size of the table, and the amount of free space remaining in the table. The table may be over-provisioned, which allows additional Config Blocks to be added after the table's initial creation.

The size of all Config Blocks must be an even multiple of 32-bits as all Config Blocks are DWORD aligned. When enumerating a Config Block Table one may assume that all Config Blocks are packed back-to-back, hence the enumeration algorithm can use the length of each config block to find the next config block in the table.

Each Config Block has a GUID that identifies which C structure the Config Block contains. All Config Blocks start with a standardized `CONFIG_BLOCK_HEADER`, which allows all the structures in the table to be enumerated even if the format of the data payload inside each Config Block is unknown.

```
00
typedef struct _CONFIG_BLOCK_HEADER {
    EFI_HOB_GUID_TYPE GuidHob; ///< Offset 0-23 GUID extension HOB header
    UINT8 Revision; ///< Offset 24 Revision of this config block
    UINT8 Attributes; ///< Offset 25 The main revision for config block
    UINT8 Reserved[2]; ///< Offset 26-27 Reserved for future use
} CONFIG_BLOCK_HEADER;
```

The `CONFIG_BLOCK_HEADER` uses the header of a GUID HOB to store its length and GUID. This makes it easy to copy Config Blocks to the HOB list. Copying Config Blocks to the HOB list is a common use case. Often the ACPI code uses policies that were initially supplied by a Config Block. Copying the Config Block to the HOB list makes it easy for DXE phase to copy any needed policies to the ACPI global NVS memory later.

The Revision field assists with binary backward compatibility. Whenever a new field is added to a Config Block, the new field is added to the bottom of the Config Block and the Revision is incremented by 1. Thus if a bootloader wishes to retain compatibility with new and old versions of the FSP binary, it may do so by either by only using fields that are present in Revision 1 of the Config Block or by checking the Revision number before modifying fields added since Revision 1.

The `CONFIG_BLOCK_TABLE_HEADER` builds upon the `CONFIG_BLOCK_HEADER`:

```
00
typedef struct _CONFIG_BLOCK_TABLE_STRUCT {
```

```
CONFIG_BLOCK_HEADER Header; ///< Offset 0-27 GUID number for main entry of config block
UINT8 Rsvd0[2]; ///< Offset 28-29 Reserved for future use
UINT16 NumberOfBlocks; ///< Offset 30-31 Number of config blocks (N)
UINT32 AvailableSize; ///< Offset 32-35 Current config block table size
///<
///< Individual Config Block Structures are added starting here
///<
} CONFIG_BLOCK_TABLE_HEADER;
```

4.3.3 Config Block Library APIs

To assist in the creation and parsing of Config Blocks, Intel has provided an open source Config Block library in [IntelSiliconPkg/Library/BaseConfigBlockLib](#). This library provides the following functions:

- GetConfigBlock()
- AddConfigBlock()
- CreateConfigBlockTable()

In most cases, bootloaders will only need to call GetConfigBlock().

4.3.4 Config Blocks used by FSP-M

On **Kaseyville** platforms SI_PREMEM_POLICY_PPI contains a Config Block Table. SI_PREMEM_POLICY_PPI will construct this Config Block Table with all the below Config Blocks pre-populated. Additionally, all the Config Blocks will be initialized with default policy values as described in section 4.1.

SI_PREMEM_POLICY_PPI contains the following Config Blocks:

- SI_PRE_MEM_CONFIG
- SMBUS_PREMEM_CONFIG
- PCH_GENERAL_PREMEM_CONFIG
- LPC_PREMEM_CONFIG
- CPU_POWER_MANAGEMENT_VR_CONFIG

4.3.5 Config Blocks used by FSP-S

On **Kaseyville** platforms SI_POLICY_PPI contains a Config Block Table. SI_POLICY_PPI will construct this Config Block Table with all the below Config Blocks pre-populated. Additionally, all the Config Blocks will be initialized with default policy values as described in section 4.1.

SI_POLICY_PPI contains the following Config Blocks:

- SI_CONFIG
 - FLASH_PROTECTION_CONFIG
 - IO_APIC_CONFIG
 - PM_CONFIG
 - ESPI_CONFIG
 - PCH_GENERAL_CONFIG
 - LOCKDOWN_CONFIG
-

- ADR_CONFIG
- INTERRUPT_CONFIG
- IBL_SOC_CONFIG
- LPSS_UART_CONFIG
- RTC_CONFIG
- P2SB_CONFIG
- SPI_CONFIG
- CPU_POWERMANAGEMENT_BASIC_CONFIG
- CPU_POWERMANAGEMENT_CUSTOM_CONFIG
- CPU_POWERMANAGEMENT_TEST_CONFIG

4.4 FSP Error Information

In the case of a fatal error occurring during the execution of the FSP, it may not be possible for the FSP to continue.

If a fatal error that prevents the successful completion of the FSP occurs, the FSP may use FSP_ERROR_INFO to report this error to the bootloader. During PEI phase, (*PeiServices)->ReportStatusCode() shall be used to transmit this error notification to the bootloader. During DXE phase, EFI_STATUS_CODE_PROTOCOL.ReportStatusCode() shall be used to transmit this error notification to the bootloader. The bootloader must ensure that ReportStatusCode() services are available before FSP-M begins execution.

```
00
typedef struct {
    EFI_STATUS_CODE_DATA DataHeader;
    EFI_GUID ErrorType;
    EFI_STATUS Status;
} FSP_ERROR_INFO;
```

FSP_ERROR_INFO is provided as the optional EFI_STATUS_CODE_DATA parameter to ReportStatusCode(). EFI_STATUS_CODE_DATA provides a CallerId GUID, this CallerId combined with the ErrorType GUID describes the error to the bootloader. The FSP for this platform implements the following CallerId GUIDs:

CallerId	Description
{ 0x5a47c211, 0x642f, 0x4f92, { 0x9c, 0xb3, 0x7f, 0xeb, 0x93, 0xda, 0xdd, 0xba}}	gMrcFspErrorType← CallerId

If the CallerId parameter is not a GUID in the table above, then it should be the GUID that identifies the PEIM or DXE driver which was executing at the time of the error.

The following ErrorType GUIDs are implemented:

ErrorType	Description
{ 0x5de1c071, 0x2c9c, 0x4a53, { 0x80, 0x21, 0x4e, 0x80, 0xd2, 0x5d, 0x44, 0xa8}}	gMrcFspErrorTypeMemoryInit

When the FSP calls ReportStatusCode(), the Type parameter's EFI_STATUS_CODE_TYPE_MASK must be EFI←
_ERROR_CODE with the EFI_STATUS_CODE_SEVERITY_MASK >= EFI_ERROR_UNRECOVERED. The Value and Instance parameters must be 0.

The bootloader must register a listener for this status code. This listener should check if DataHeader.Type == STATUS_CODE_DATA_TYPE_FSP_ERROR_GUID to detect an FSP_ERROR_INFO notification. If an FSP_E←
RROR_INFO notification is encountered, the bootloader should assume that normal operation is no longer possible. In debug scenarios, this notification should be considered an ASSERT.

4.5 Dispatch Mode Integration

Dispatch Mode Integration Notes:

1. The FSP for this platform contains PEIMs compiled for the IA32 architecture. The boot loader therefore must utilize a PEI Foundation compiled for the IA32 architecture.
2. Since the FSP binary can be integrated into flash at any address, the boot loader has to report FSP FVs to the PEI and DXE dispatcher using PI specification defined mechanisms so PEIMs and DXE drivers inside the FSP Binary can be dispatched. FspmWrapperPeim and FspWrapperPeim from IntelFsp2WrapperPkg can aid in implementing this.
3. For this platform, FSP-T, FSP-M and FSP-S contain 1 FV each.
4. The FSP distribution package will include a DSC file which contains all DynamicEx PCDs consumed by the FSP binary. The boot loader should include the DSC during its build process so that any PCDs defined by this DSC file are included in the boot loader's PCD database. This enables the boot loader and FSP to share a single PCD database.

- A NULL library (FspPcdListLibNull.inf) is included in the FSP distribution package. This library should be included in one of the boot loader's PEIMs. This ensures all DynamicEx PCDs used by the FSP are included in the boot loader's PCD database. One can fulfill this requirement by including the following code snippet in *BoardPkg.dsc:

```
00
IntelFsp2WrapperPkg/FspmWrapperPeim/FspmWrapperPeim.inf {
  <LibraryClasses>
  !if gIntelFsp2WrapperTokenSpaceGuid.PcdFspModeSelection == 0
  #
  # In FSP Dispatch mode below dummy library should be linked to bootloader PEIM
  # to build all DynamicEx PCDs that FSP consumes into bootloader PCD database.
  #
  NULL|$(PLATFORM_FSP_BIN_PACKAGE)/Library/FspPcdListLib/FspPcdListLibNull.inf
  !endif
}
```

5. The boot loader must provide at minimum 128KB of stack and 256KB of HOB heap to execute FSP on this platform.
6. In dispatch mode, the boot loader should not use FSP API calls described in chapter 5 of this document or chapter 10 of the FSP External Architecture Specification version 2.4. The TempRamInit API is the only exception, it is supported in both API mode and dispatch mode. All other APIs (MemoryInit, SiliconInit, etc.) should not be invoked.
7. For dispatch mode, FSP contains x64 DXE drivers to replace the NotifyPhase API. This eliminates thunking from 64bit to 32bit when using FSP dispatch mode. The boot loader should remove S3EndOfPeiNotify and FspWrapperNotifyDxe since they are no longer used in dispatch mode.
8. EFI_PEI_CORE_FV_LOCATION_PPI should be installed by the boot loader's SEC phase. EFI_PEI_CORE_FV_LOCATION_PPI.PeiCoreFvLocation should point to the first Firmware Volume (FV) in FSP-M so the PeiCore inside FSP will be invoked. If EFI_PEI_CORE_FV_LOCATION_PPI is not installed or PeiCore cannot be found at the address specified by EFI_PEI_CORE_FV_LOCATION_PPI.PeiCoreFvLocation, the PeiCore from the Boot Firmware Volume (BFV) will be invoked instead.
9. FSP-S requires multi-threaded code to complete silicon initialization on this platform. FSP-S includes the UefiCpuPkg/CpuMpPei/CpuMpPei.inf PEIM to provide multiprocessing. The bootloader can choose to either use the MP_SERVICES provided by the FSP for all PEIMs or the bootloader may provide an alternative implementation. In either case, only one MP_SERVICES implementation can be active at once. If the bootloader wishes to not use the FSP provided MP_SERVICES, then the bootloader must install the MP_SERVICES_PPI before installing the FV_INFO_PPI for FSP-S. If MP_SERVICES_PPI already exists, then FSP-S will use it and not execute its own implementation.
10. FSPM_ARCH_CONFIG_PPI->NvsBufferPtr is now a universal policy option (FSP Dispatch Mode and EDK2 native.) To enable the fast MRC training flow, the boot loader or platform code must to install this PPI to restore the previous MRC training data (SA_MISC_PEI_PREMEM_CONFIG->S3DataPtr is obsolete).
11. Debug message handling in dispatch mode:

- Before the ReportStatusCode service is ready, a debug built FSP will send debug messages using the FSP-T UPD configuration (passed as FSP-T API input parameter). FSP-T is recommended to be used regardless FSP API mode or Dispatch mode.
 - Once the ReportStatusCode service is ready, a debug built FSP will send debug messages using the ReportStatusCode service.
 - It is recommended that bootloader register a StatusCode listener immediately after the ReportStatusCode service is ready. It is important to register this listener as soon as possible so that all debug messages sent by the FSP are captured.
 - Please refer to section 10.4.12 in the Intel(R) Firmware Support Package External Architecture Specification v2.4 for details about the ReportStatusCode debug message format.
-

Chapter 5

FSP API Mode

5.1 Overview

This release of the FSP supports the all APIs required by the FSP External Architecture Specification version 2.4. These APIs are only used when running the FSP in API mode. In Dispatch mode, these APIs are not used (with the exception of TempRamInit.) The FSP information header contains the address offset for these APIs. Register usage is described in the FSP External Architecture Specification version 2.4. Any usage not described by the specification is described in the individual sections below.

The sections below will highlight any changes that are specific to this FSP release.

5.2 FSP APIs

5.2.1 TempRamInit API

Please refer Chapter 9.7 in the FSP External Architecture Specification version 2.4 for complete details including the prototype, parameters and return value details for this API.

TempRamInit does basic early initialization primarily setting up temporary RAM using cache. It returns ECX pointing to beginning of temporary memory and EDX pointing to end of temporary memory + 1. The total temporary ram currently available is given by PcdTemporaryRamSize starting from the base address of PcdTemporaryRamBase. Out of the total temporary memory available, the last PcdFspReservedBufferSize bytes of space are reserved by the FSP for TempRamInit if temporary RAM initialization is done by the FSP. Any remaining space from **TemporaryRamBase(ECX)** to **TemporaryRamBase+TemporaryRamSize-FspReservedBufferSize** (EDX) is available for both bootloader and FSP use.

It is a requirement for Firmware to have a Firmware Interface Table (FIT). The FIT contains pointers to each microcode update. The microcode update is loaded for all logical processors before executing the reset vector. If more than one microcode update for the CPU is present, the microcode update with the latest revision is loaded.

FSPT_UPD.MicrocodeRegionBase** and **FSPT_UPD.MicrocodeRegionLength** are input parameters to TempRamInit API. If these values are 0, FSP will not attempt to update microcode. If MicrocodeRegionBase != 0 and MicrocodeRegionLength != 0, then FSP will search that region for microcode updates. If a newer microcode update revision is found in the region, FSP will load it.

5.2.2 FspMemoryInit API

Please refer to Chapter 9.8 in the FSP External Architecture Specification version 2.4 for the prototype, parameters and return value details for this API.

The variable **FspmUpdPtr** is a pointer to the **FSPM_UPD** structure which is described in the header file [FspmUpd.h](#).

The bootloader must pass valid a CAR region for FSP through **FSPM_UPD.FspmArchUpd.StackBase** and **FSP_M_UPD.FspmArchUpd.StackSize** UPDs.

The FSP for this platform will run `FspMemoryInit` top of the stack provided by the bootloader instead of establishing a separate stack as described by the FSP External Architecture Specification version 2.4. The memory region provided by the **FSPM_UPD.FspmArchUpd.StackBase** and **FSPM_UPD.FspmArchUpd.StackSize** UPDs is used to establish a HOB heap. The names **StackBase** and **StackSize** can be confusing since they are **NOT** used for stack. These names were retained for backwards compatibility with FSP v2.0. Stack requirement: FSP's stack usage starts from the current stack pointer. The minimum stack size requirement for FSP-M is 256KB, stack base is 0xFE17F00 by default.

The bootloader must ensure that sufficient stack space is available to fulfill the FSP-M minimum stack size requirement at the point in execution where `FspMemoryInit()` is called. The stack allocated by the bootloader must be large enough for both FSP-M as well as any other parent function calls that are still on the stack at the point when `FspMemoryInit()` is called.

After `FspMemoryInit()` is completed, permanent memory is available. After this point, the memory pressure experienced early in boot is eliminated. Accordingly, right before `FspMemoryInit()` exits, any data that needs to be retained for later use by `FspSiliconInit()` will be copied to permanent memory. `FspSiliconInit()` will then execute on a second stack.

The base address of HECI device (Bus 0, Device 22, Function 0) is required to be initialized prior to calling `FspMemoryInit()`. The default address is programmed to 0xFED1A000.

`FspMemoryInit()` will calculate the memory map by taking into account the size of several memory regions: TSEG, GTT, BDSM, ME stolen, Uncore PMRR, IOT, MOT, DPR, REMAP, TOLUD, TOUUD. These memory regions may not be initialized by `FspMemoryInit()`, but space will be reserved for them.

5.2.3 TempRamExit API

Please refer to Chapter 9.9 in the FSP external Architecture Specification version 2.4 for the prototype, parameters and return value details for this API.

If Boot Loader initializes the Temporary RAM (CAR) and skip calling **TempRamInit API**, it is expected that bootloader must skip calling this API and bootloader will tear down the temporary memory area setup in the cache and bring the cache to normal mode of operation.

The FSP for this platform doesn't have any input parameters for this API. The value of *TempRamExitParamPtr* should be NULL.

At the end of *TempRamExit* the original code and data cache are disabled. FSP will reconfigure all MTRRs. If the boot loader wishes to configure the MTRRs differently, they can be reprogrammed immediately after this API call.

5.2.4 FspSiliconInit API

Please refer to Chapter 9.10 in the FSP external Architecture Specification version 2.4 for the prototype, parameters and return value details for this API.

The variable *FspsUpdPtr* is a pointer to the **FSPS_UPD** structure which is described in the header file [FspsUpd.h](#).

It is expected that the boot loader will adjust MTRRs for SBSP if needed after **TempRamExit** but before entering **FspSiliconInit**. If the MTRRs are not programmed properly, boot performance can be impacted.

The region of 0x5_8000 - 0x5_8FFF is used by `FspSiliconInit` for starting APs. If this data is important to bootloader, then bootloader needs to preserve it before calling `FspSiliconInit`.

FspSiliconInit requires multi-threaded code to complete silicon initialization on this platform. FSP includes the `UefiCpuPkg/CpuMpPei/CpuMpPei.inf` PEIM to provide multiprocessing. Accordingly, the boot loader must expect FSP to perform an INIT-SIPI-SIPI during **FspSiliconInit** and **NotifyPhase** which will take control of all APs in the system and restart them. This will kill any background threads that were running on the APs. In some cases, this may not be desirable. As an alternative, the boot loader may provide an instance of the `EFI_PEI_MP_P_SERVICES_PPI` through the `FspsUpd->FspsConfig.CpuMpPpi` UPD for the FSP to use instead of the built-in implementation. **This is entirely optional**, if callbacks from FSP to the boot loader are not desired, one may

```
set FspUpd->FspConfig.CpuMpPpi == NULL.
```

In summary, there are two methods that FSP can use for performing multiprocessing:

1. Using the multiprocessing services built-in to the FSP.
2. Using the boot loader's multiprocessing services.

Using method #1, the boot loader will set `FspUpd->FspConfig.CpuMpPpi == NULL`. If this UPD is `NULL`, then FSP will use its built-in MP services implementation for multiprocessing. Accordingly, the boot loader must expect FSP to perform an INIT-SIPI-SIPI during **FspSiliconInit** and **NotifyPhase** which will take control of all APs in the system and restart them. This will kill any background threads that were running on the APs.

Using method #2, the boot loader will set `FspUpd->FspConfig.CpuMpPpi != NULL`. If this UPD is not `NULL`, it must point to an instance of `EFI_PEI_MP_SERVICES_PPI`. When the FSP needs to perform multiprocessing, it will use the `EFI_PEI_MP_SERVICES_PPI` instance provided by the boot loader to do so. Accordingly, the boot loader must expect the function pointers in `EFI_PEI_MP_SERVICES_PPI` to be invoked in the middle of the execution of **FspSiliconInit** and **NotifyPhase**.

Note

If the boot loader is a UEFI boot loader using API mode instead of dispatch mode, and `FspUpd->FspConfig.CpuMpPpi != NULL`, then `FspUpd->FspConfig.CpuMpHob` must be `!= NULL`. This UPD is a pointer to the `CPU_MP_DATA` HOB.

It is a requirement for the bootloader to have a Firmware Interface Table (FIT), which contains pointers to each microcode. The microcode is loaded for all cores before reset vector. If more than one microcode update for the CPU is present, the latest revision is loaded.

Stack requirement: 4KB of free stack space should be provided to execute *FspSiliconInit*.

5.2.5 NotifyPhase API

Please refer Chapter 9.13 in the FSP External Architecture Specification version 2.4 for the prototype, parameters and return value details for this API.

Stack requirement: 4KB of free stack space should be provided to execute *NotifyPhase*.

5.2.5.1 PostPciEnumeration Notification

This phase *EnumInitPhaseAfterPciEnumeration* is to be called after PCI enumeration but before execution of third party code such as option ROMs. It includes powering down unused PCH SATA ports, locking registers in PMC, DMI, TCO, and SPI Flash (DLOCK + WRSDIS + FLOCKDN). It also includes enabling bus mastering for eSPI connected devices.

5.2.5.2 ReadyToBoot Notification

This phase *EnumInitPhaseReadyToBoot* is to be called before giving control to the operating system. It includes some final initialization steps recommended by the BWG, including power management settings, and sending ME Message EOP (End of Post).

5.2.5.3 EndOfFirmware Notification

This phase *EnumInitEndOfFirmware* is to be called before the firmware/preboot environment transfers management of all system resources to the OS or next level execution environment. It includes final locking of chipset registers.

5.3 Reset Return Codes

As per FSP External Architecture Specification version 2.4, any reset required in the FSP flow will be reported by returning one of the FSP_STATUS_RESET_REQUIRED* return codes.

It is the boot loader's responsibility to reset the system according to the reset type requested.

Below table specifies the return status returned by FSP API and the requested reset type.

FSP_STATUS_RESET_REQUIRED Code	Reset Type requested
0x40000001	Cold Reset
0x40000002	Warm Reset
0x40000003	Global Reset - Puts the system through a Global Reset through HECI or a Full Reset through PCH
0x40000004	Reserved
0x40000005	Reserved
0x40000006	Reserved
0x40000007	Reserved
0x40000008	Reserved

5.4 UPD Porting Guide

Recommended values for UPDs:

Base on requirement to provide UPD recommendation value. For now, no UPD requirement received yet.

Chapter 6

Porting Recommendations

Here are some notes and recommendations for adapting an existing boot loader to FSP.

6.1 Locking PAM register

FSP 2.0 introduced the EndOfFirmware Notify phase callback which is the recommended place for locking PAM registers. Accordingly, by default FSP locks the PAM registers during the EndOfFirmware Notify phase. If the EndOfFirmware Notify phase is too early to lock PAM registers, then the PAM locking code inside the FSP can be disabled by setting the UPD -> FSP_S_TEST_CONFIG -> SkipPamLock or SA policy -> _SI_PREMEM_POLICY_STRUCT -> SA_MISC_PEI_CONFIG -> SkipPamLock. If the PAM locking code inside the FSP is skipped, then the boot loader must lock the PAM registers before booting the OS. by programming one PCI config space register as below.

The PAM registers must be locked in all boot paths including S3 resume. To lock the PAM registers, program the following lock bit:

```
00
MmioOr32 (B0: D0: F0: Register 0x80, BIT0)
```

6.2 Locking SMRAM register

It is recommended that SMRAM be locked before any third party code (e.g. OpROM) execution. The point in execution where third party OpROMs begin executing can vary depending on the boot loader implementation. To provide flexibility, the FSP by default will not lock it. The boot loader should lock SMRAM by programming the following lock bit before any third party OpROM execution. Additionally, SMRAM should be locked before the **EnumInitPhaseReadyToBoot** notify phase is called. During S3 resume, the lock bit should be set right before the OS wake vector.

```
00
PciOr8 (B0: D0: F0: Register 0x88, BIT4);
Note: This register must be programmed using the legacy CF8/CFC PCI access mechanism. (MMIO access will
not work)
```

6.3 Locking SMI register

It is recommended that the global SMI bit is locked before any third party code (e.g. OpROM) execution. SMM initialization flows may vary depending on boot loader implementation details. Accordingly, FSP will not lock it by default. The boot loader is responsible for locking the following registers after SMM configuration is complete. Set AcpiBase + 0x30[0] to 1b to enable global SMI. Set PMC PCI offset A0h[4] = 1b to lock SMI.

6.4 Verify below settings are correct for your platforms

PMC PCI configuration space is not PCI specification compliant. The FSP will hide the PMC controller to avoid external software or OS from corrupting the BAR addresses. FSP will program the PMC controller IO and MIO BAR's with below addresses. Please use these addresses in the boot loader code instead of reading BAR addresses from the PMC controller.

Register	Values
ABASE	0x1800
PWRMBASE	0xFE000000
PCIEXBAR_BASE_ADDRESS	0xE0000000

Note

:

- Boot Loader can use a different value for PCIEXBAR_BASE_ADDRESS either by modifying the UPD (under FSP-T) or by overriding the PCIEXBAR (B0:D0:F0:R60h) before calling FspMemoryInit.
 - Boot Loader should avoid using conflicting address when reprogramming PCIEXBAR_BASE_ADDRESS than the recommended one.
-

Chapter 7

FSP Output

The FSP builds a series of data structures called Hand-Off-Blocks (HOBs) as it progresses through initializing the silicon.

Please refer to the Platform Initialization (PI) Specification - Volume 3: Shared Architectural Elements specification for PI Architectural HOBs. Please refer Chapter 11 in the FSP External Architecture Specification version 2.4 for details about FSP Architectural HOBs.

The sections below describe the HOBs not covered in the above two specifications.

7.1 FSP_ERROR_INFO_HOB

In the case of an error occurring during the execution of the FSP, the FSP may produce this architectural HOB which describes the error in more detail. [FSP_ERROR_INFO_HOB](#) is only used in FSP API Mode. In FSP Dispatch Mode, FSP may call ReportStatusCode() and provide a FSP_ERROR_INFO structure using the PI status code services.

- [FspErrorInfoHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{0x611e6a88, 0xad7, 0x4301, { 0x93, 0xff, 0xe4, 0x73, 0x04, 0xb4, 0x3d, 0xa6 }}	FSP_ERROR_INFO_HOB_GUID

The FSP for this platform implements the following CallerId GUIDs:

CallerId	Description
{ 0x5a47c211, 0x642f, 0x4f92, { 0x9c, 0xb3, 0x7f, 0xeb, 0x93, 0xda, 0xdd, 0xba }}	gMrcFspErrorType↔ CallerId

The following ErrorType GUIDs are implemented:

ErrorType	Description
{ 0x5de1c071, 0x2c9c, 0x4a53, { 0x80, 0x21, 0x4e, 0x80, 0xd2, 0x5d, 0x44, 0xa8 }}	gMrcFspErrorTypeMemoryInit

7.2 CXL_CEDT_HOB

This HOB contains information necessary to populate the ACPI CEDT. The existence of this HOB is optional and dependent on the platform configuration.

- [CxlCedtHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0x58F943E7, 0xBB51, 0x445F, { 0xB9, 0x81, 0xF8, 0xBE, 0xBB, 0x0A, 0x30, 0x1B } }	CXL_CEDT_HOB_GUID

7.3 CXL_NODE_SOCKET

In API Mode, FSP may produce this HOB to describe CXL Node information in more detail. The information in this HOB can be consumed by bootloader to perform further platform settings. The existence of this HOB is dependent on platform hardware configuration. Upon successful discovery of this HOB from the HOB list, the bootloader may use the following cast to access the HOB data: CXL_NODE_SOCKET

- [CxlNodeHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0xdd8ae009, 0xda5a, 0x44a3, { 0xbe, 0x18, 0xda, 0x0c, 0x16, 0xc5, 0xaf, 0x5c } }	CXL_NODE_HOB_GUID

7.4 IIO_UDS

This HOB provides information on the initialized IIO topology performed inside FSP API calls.

- [IioUniversalDataHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0xa1, 0x96, 0xf3, 0x7f, 0x7d, 0xee, 0x1e, 0x43, 0xba, 0x53, 0x8f, 0xca, 0x12, 0x7c, 0x44, 0xc0 }	IIO_UNIVERSAL_DATA_GUID

7.5 SYSTEM_MEMORY_MAP_HOB

- [MemoryMapDataHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0xf8870015, 0x6994, 0x4b98, 0x95, 0xa2, 0xbd, 0x56, 0xda, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }	MEMORY_MAP_HOB_GUID

7.6 PREV_BOOT_ERR_SRC_HOB

This HOB provides information on error sources reported during the previous boots.

- [PrevBootErrSrcHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0xc5, 0xb5, 0x38, 0x51, 0x69, 0x93, 0xec, 0x48, 0x5b, 0x97, 0x38, 0xa2, 0xf7, 0x09, 0x66, 0x75 }	FSP_PREV_BOOT_ERR_SRC_HOB_GUID

7.7 SYSTEM_INFO_VAR

This HOB provides system information collected through silicon initialization.

- [SystemInfoHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0x7650A0F2, 0x0D91, 0x4B0C, { 0x92, 0x3B, 0xBD, 0xCF, 0x22, 0xD1, 0x64, 0x35 } }	SYSTEM_INFO_HOB_GUID

7.8 FSP_EXT_MEMORY_PPR_HOB

This HOB provides information on memory PPR (Post-Package Repair) used to identify damaged DIMMs.

- [FspExtMemoryPprHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0x3956C6DA, 0x35B6, 0x4036, { 0x93, 0xE4, 0xB1, 0x51, 0x38, 0x60, 0x21, 0x1E } }	FSP_EXT_MEMORY_PPR_HOB_GUID

7.9 FSP_EXT_SYSTEM_MEMORY_MAP_HOB

This HOB provides information that describes the system memory map collected through silicon initialization.

- [FspExtSystemMemoryMapHob.h](#)

The FSP for this platform implements this HOB with the following GUID:

HOB GUID	Macro
{ 0xDF310DE8, 0x579F, 0x419C, { 0x↔B6, 0xAB, 0x4D, 0x4B, 0xE7, 0xCA, 0xB0, 0x83 } }	FSP_EXT_SYSTEM_MEMORY_MAP_HOB_GUID

Chapter 8

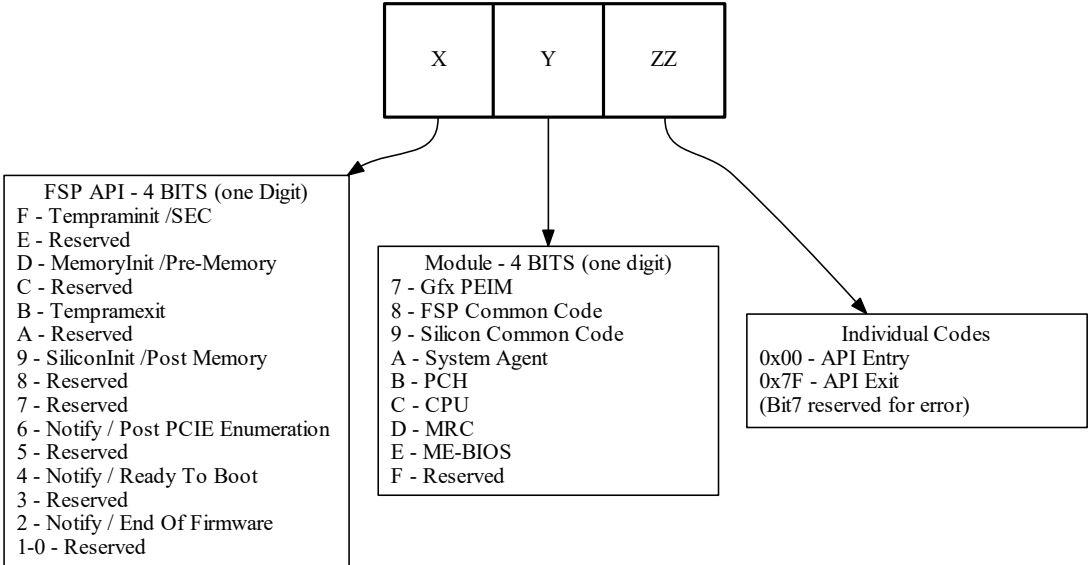
POST Codes

The FSP outputs 16-bit POST codes to indicate which API and which module is currently executing.

Bit Range	Description
Bit15 - Bit12 (X)	used to indicate the phase/api under which the code is executing
Bit11 - Bit8 (Y)	used to indicate the module
Bit7 (ZZ bit 7)	reserved for error
Bit6 - Bit0 (ZZ)	individual codes

8.1 POST Code Info

The diagram below illustrates how sub-fields are encoded in the POST codes produced by the FSP.



8.1.1 TempRamInit API Status Codes (0xFxxx)

8.1.2 FspMemoryInit API Status Codes (0xDxxx)

8.1.3 TempRamExit API Status Codes (0xBxxx)

8.1.4 FspSiliconInit API Status Codes (0x9xxx)

8.1.5 NotifyPhase API Status Codes (0x6xxx)

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- [CEDT_CFMWS_WINDOW_RESTRICTIONS](#)
CEDT CXL Fixed Memory Window Structure (CFMWS) Window Restrictions 31
- [CXL_CEDT_HOB](#)
CXL CEDT HOB data structure 31
- [FSP_ERROR_INFO_HOB](#)
FSP Error Information Block 31
- [PLATFORM_DATA](#)
This structure keeps resource ranges configured in whole system 32
- [UDS_PCIROOT_RES](#)
PCI resources that establish one PCI hierarchy for PCI Enumerator 33
- [UDS_SOCKET_RES](#)
This structure keeps resource ranges configured in one socket 34
- [UDS_STACK_RES](#)
This structure keeps resources configured in Host I/O Processor (HIOP) for one stack 34

Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

CxlCedtHob.h	CXL CEDT GUID HOB definitions	37
CxlNodeHob.h	CXL GUID HOB definitions	37
FspAcpiHobs.h	RAS ACPI GUID HOB definitions	38
FspEdpcParam.h	RAS EDPC PARAMETER Structure definitions	38
FspErrorInfoHob.h	FSP Error Information HOB to describe errors inside FSP that bootloader may take some actions to handle those error scenarios	39
FspExtMemoryPprHob.h	FSP Output HOB: MemoryPpr	40
FspExtSystemMemoryMapHob.h	FSP Output HOB: SystemMemoryMap	40
FspGlobals.h	FSP Output HOB: SystemMemoryMap	41
FspiUpd.h	Platform Configuration C Struct Header File	42
FspmUpd.h	Platform Configuration C Struct Header File	43
FspSUpd.h	Platform Configuration C Struct Header File	44
FspTUpd.h	Platform Configuration C Struct Header File	44
FspUpd.h	Platform Configuration C Struct Header File	45
lioPcieConfigUpd.h	Header file to define the structures used for FSPM dynamic configuration	46
lioUniversalDataHob.h	Data format for IIO Universal Data HOB Structure	47
MemoryMapDataHob.h	GUID used for Memory Map Data entries in the HOB list	47
PrevBootErrSrcHob.h	Previous Boot Error Source HOB Header File	48
SystemInfoHob.h	System Information HOB Header File	49

Chapter 11

Class Documentation

11.1 CEDT_CFMWS_WINDOW_RESTRICTIONS Union Reference

CEDT CXL Fixed Memory Window Structure (CFMWS) Window Restrictions.

```
#include <CxlCedtHob.h>
```

11.1.1 Detailed Description

CEDT CXL Fixed Memory Window Structure (CFMWS) Window Restrictions.

Definition at line 84 of file CxlCedtHob.h.

The documentation for this union was generated from the following file:

- [CxlCedtHob.h](#)

11.2 CXL_CEDT_HOB Struct Reference

CXL CEDT HOB data structure.

```
#include <CxlCedtHob.h>
```

11.2.1 Detailed Description

CXL CEDT HOB data structure.

Definition at line 139 of file CxlCedtHob.h.

The documentation for this struct was generated from the following file:

- [CxlCedtHob.h](#)

11.3 FSP_ERROR_INFO_HOB Struct Reference

FSP Error Information Block.

```
#include <FspErrorInfoHob.h>
```

Public Attributes

- `EFI_HOB_GUID_TYPE` [GuidHob](#)

GUID HOB header.

- `EFI_STATUS_CODE_TYPE` [Type](#)

ReportStatusCode () type identifier.

- `EFI_STATUS_CODE_VALUE` [Value](#)

ReportStatusCode () value.

- `UINT32` [Instance](#)

ReportStatusCode () Instance number.

- `EFI_GUID` [CallerId](#)

Optional GUID which may be used to identify which internal component of the FSP was executing at the time of the error.

- `EFI_GUID` [ErrorType](#)

GUID identifying the nature of the fatal error.

- `UINT32` [Status](#)

EFI_STATUS code describing the error encountered.

11.3.1 Detailed Description

FSP Error Information Block.

Definition at line 36 of file `FspErrorInfoHob.h`.

The documentation for this struct was generated from the following file:

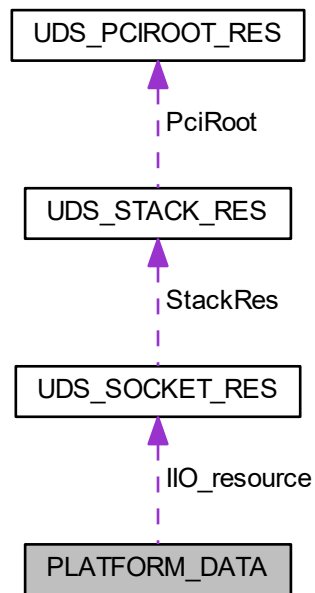
- [FspErrorInfoHob.h](#)

11.4 PLATFORM_DATA Struct Reference

This structure keeps resource ranges configured in whole system.

```
#include <IioUniversalDataHob.h>
```

Collaboration diagram for PLATFORM_DATA:



11.4.1 Detailed Description

This structure keeps resource ranges configured in whole system.

Definition at line 167 of file IioUniversalDataHob.h.

The documentation for this struct was generated from the following file:

- [IioUniversalDataHob.h](#)

11.5 UDS_PCIROOT_RES Struct Reference

PCI resources that establish one PCI hierarchy for PCI Enumerator.

```
#include <IioUniversalDataHob.h>
```

11.5.1 Detailed Description

PCI resources that establish one PCI hierarchy for PCI Enumerator.

Definition at line 111 of file IioUniversalDataHob.h.

The documentation for this struct was generated from the following file:

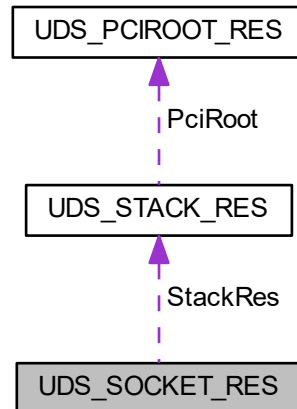
- [IioUniversalDataHob.h](#)
-

11.6 UDS_SOCKET_RES Struct Reference

This structure keeps resource ranges configured in one socket.

```
#include <IioUniversalDataHob.h>
```

Collaboration diagram for UDS_SOCKET_RES:



11.6.1 Detailed Description

This structure keeps resource ranges configured in one socket.

It contains a table of IO stacks provided by the socket. The stacks are also grouped by IO dies, but dies are not reflected in UDS.

Definition at line 152 of file IioUniversalDataHob.h.

The documentation for this struct was generated from the following file:

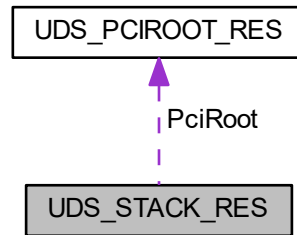
- [IioUniversalDataHob.h](#)

11.7 UDS_STACK_RES Struct Reference

This structure keeps resources configured in Host I/O Processor (HIOP) for one stack.

```
#include <IioUniversalDataHob.h>
```

Collaboration diagram for UDS_STACK_RES:



11.7.1 Detailed Description

This structure keeps resources configured in Host I/O Processor (HIOP) for one stack.

One HIOP may produce more than one PCI hierarchy, these are in PciRoot[] table.

Definition at line 127 of file lioUniversalDataHob.h.

The documentation for this struct was generated from the following file:

- [lioUniversalDataHob.h](#)

Chapter 12

File Documentation

12.1 CxlCedtHob.h File Reference

CXL CEDT GUID HOB definitions.

Classes

- union [CEDT_CFMWS_WINDOW_RESTRICTIONS](#)
CEDT CXL Fixed Memory Window Structure (CFMWS) Window Restrictions.
- struct [CXL_CEDT_HOB](#)
CXL CEDT HOB data structure.

12.1.1 Detailed Description

CXL CEDT GUID HOB definitions.

Copyright

INTEL CONFIDENTIAL Copyright 2022 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.2 CxlNodeHob.h File Reference

CXL GUID HOB definitions.

12.2.1 Detailed Description

CXL GUID HOB definitions.

Copyright

INTEL CONFIDENTIAL Copyright 2021 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.3 FspAcpiHobs.h File Reference

RAS ACPI GUID HOB definitions.

12.3.1 Detailed Description

RAS ACPI GUID HOB definitions.

Copyright

INTEL CONFIDENTIAL Copyright 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.4 FspEdpcParam.h File Reference

RAS EDPC PARAMETER Structure definitions.

12.4.1 Detailed Description

RAS EDPC PARAMETER Structure definitions.

Copyright

INTEL CONFIDENTIAL Copyright 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.5 FspErrorInfoHob.h File Reference

FSP Error Information HOB to describe errors inside FSP that bootloader may take some actions to handle those error scenarios.

Classes

- struct [FSP_ERROR_INFO_HOB](#)

FSP Error Information Block.

Macros

- #define [FSP_ERROR_INFO_HOB_GUID](#) { 0x611e6a88, 0xad7, 0x4301, { 0x93, 0xff, 0xe4, 0x73, 0x04, 0xb4, 0x3d, 0xa6 } }

GUID value indicating the FSP error information.

12.5.1 Detailed Description

FSP Error Information HOB to describe errors inside FSP that bootloader may take some actions to handle those error scenarios.

Copyright

INTEL CONFIDENTIAL Copyright (C) 2023 Intel Corporation.

This software and the related documents are Intel copyrighted materials, and your use of them is governed by the express license under which they were provided to you ("License"). Unless the License provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this software or the related documents without Intel's prior written permission.

This software and the related documents are provided as is, with no express or implied warranties, other than those that are expressly stated in the License.

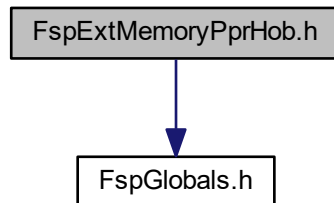
Specification Reference:

12.6 FspExtMemoryPprHob.h File Reference

FSP Output HOB: MemoryPpr.

```
#include "FspGlobals.h"
```

Include dependency graph for FspExtMemoryPprHob.h:



12.6.1 Detailed Description

FSP Output HOB: MemoryPpr.

Copyright

INTEL CONFIDENTIAL Copyright 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

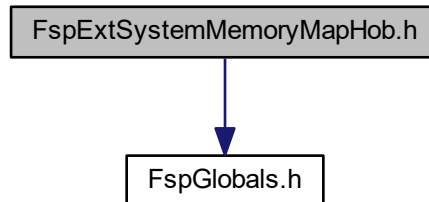
No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.7 FspExtSystemMemoryMapHob.h File Reference

FSP Output HOB: SystemMemoryMap.

```
#include "FspGlobals.h"  
Include dependency graph for FspExtSystemMemoryMapHob.h:
```



12.7.1 Detailed Description

FSP Output HOB: SystemMemoryMap.

Copyright

INTEL CONFIDENTIAL Copyright 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

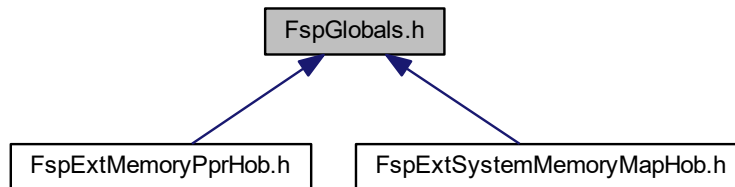
No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.8 FspGlobals.h File Reference

FSP Output HOB: SystemMemoryMap.

This graph shows which files directly or indirectly include this file:



12.8.1 Detailed Description

FSP Output HOB: SystemMemoryMap.

Copyright

INTEL CONFIDENTIAL Copyright 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

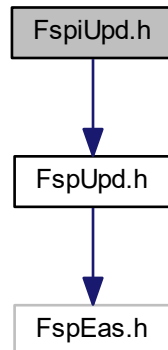
Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.9 FspiUpd.h File Reference

Platform Configuration C Struct Header File.

```
#include <FspUpd.h>
```

Include dependency graph for FspiUpd.h:



12.9.1 Detailed Description

Platform Configuration C Struct Header File.

Copyright (c) 2026, Intel Corporation. All rights reserved.

SPDX-License-Identifier: BSD-2-Clause-Patent

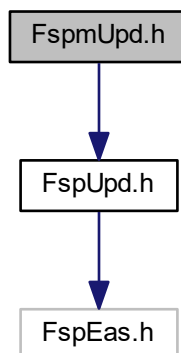
This file is automatically generated. Please do NOT modify !!!

12.10 FspmUpd.h File Reference

Platform Configuration C Struct Header File.

```
#include <FspUpd.h>
```

Include dependency graph for FspmUpd.h:



12.10.1 Detailed Description

Platform Configuration C Struct Header File.

Copyright (c) 2026, Intel Corporation. All rights reserved.

SPDX-License-Identifier: BSD-2-Clause-Patent

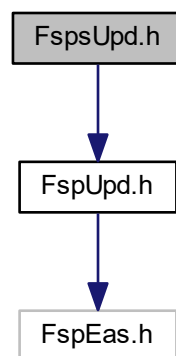
This file is automatically generated. Please do NOT modify !!!

12.11 FspUpd.h File Reference

Platform Configuration C Struct Header File.

```
#include <FspUpd.h>
```

Include dependency graph for FspUpd.h:



12.11.1 Detailed Description

Platform Configuration C Struct Header File.

Copyright (c) 2026, Intel Corporation. All rights reserved.

SPDX-License-Identifier: BSD-2-Clause-Patent

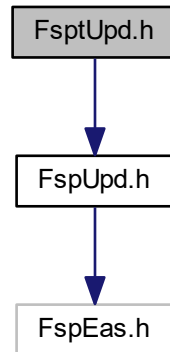
This file is automatically generated. Please do NOT modify !!!

12.12 FsptUpd.h File Reference

Platform Configuration C Struct Header File.

```
#include <FspUpd.h>
```

Include dependency graph for FspUpd.h:



12.12.1 Detailed Description

Platform Configuration C Struct Header File.

Copyright (c) 2026, Intel Corporation. All rights reserved.
SPDX-License-Identifier: BSD-2-Clause-Patent

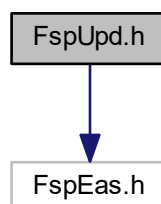
This file is automatically generated. Please do NOT modify !!!

12.13 FspUpd.h File Reference

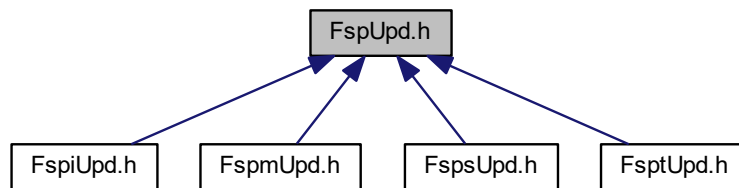
Platform Configuration C Struct Header File.

```
#include <FspEas.h>
```

Include dependency graph for FspUpd.h:



This graph shows which files directly or indirectly include this file:



12.13.1 Detailed Description

Platform Configuration C Struct Header File.

Copyright (c) 2026, Intel Corporation. All rights reserved.

SPDX-License-Identifier: BSD-2-Clause-Patent

This file is automatically generated. Please do NOT modify !!!

12.14 IioPcieConfigUpd.h File Reference

Header file to define the structures used for FSPM dynamic configuration.

Macros

- #define [MAX_IIO_PORTS_PER_STACK](#) 8
Maximum number of IIO ports per IIO stack.

12.14.1 Detailed Description

Header file to define the structures used for FSPM dynamic configuration.

Copyright

INTEL CONFIDENTIAL Copyright 2021 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission. No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing. Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.15 IioUniversalDataHob.h File Reference

Data format for IIO Universal Data HOB Structure.

Classes

- struct [UDS_PCIROOT_RES](#)
PCI resources that establish one PCI hierarchy for PCI Enumerator.
- struct [UDS_STACK_RES](#)
This structure keeps resources configured in Host I/O Processor (HIOP) for one stack.
- struct [UDS_SOCKET_RES](#)
This structure keeps resource ranges configured in one socket.
- struct [PLATFORM_DATA](#)
This structure keeps resource ranges configured in whole system.

12.15.1 Detailed Description

Data format for IIO Universal Data HOB Structure.

Copyright

INTEL CONFIDENTIAL Copyright 1999 - 2023 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.16 MemoryMapDataHob.h File Reference

GUID used for Memory Map Data entries in the HOB list.

Macros

- #define [MEM_SKTCH_TO_IMC](#)(SktCh) ((SktCh) / MAX_MC_CH)
Memory channel index conversion macros.

12.16.1 Detailed Description

GUID used for Memory Map Data entries in the HOB list.

Copyright

INTEL CONFIDENTIAL Copyright 2021 - 2022 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.16.2 Macro Definition Documentation

12.16.2.1 MEM_SKTCH_TO_IMC

```
#define MEM_SKTCH_TO_IMC(  
    SktCh ) ((SktCh) / MAX_MC_CH)
```

Memory channel index conversion macros.

We got two types of memory channel indexes:

- socket channel - indexes 0..MAX_CH, used in [socket][channel] indexing
- IMC channel - indexes 0..MAX_MC_CH, used in [socket][IMC][channel] indexing The below defined macros convert one channel index to the other one.

Definition at line 118 of file MemoryMapDataHob.h.

12.17 PrevBootErrSrcHob.h File Reference

Previous Boot Error Source HOB Header File.

12.17.1 Detailed Description

Previous Boot Error Source HOB Header File.

Copyright

INTEL CONFIDENTIAL Copyright 2007 -2021 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and

treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

12.18 SystemInfoHob.h File Reference

System Information HOB Header File.

12.18.1 Detailed Description

System Information HOB Header File.

Copyright

INTEL CONFIDENTIAL Copyright 2021 Intel Corporation.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors. The Material may contain trade secrets and proprietary and confidential information of Intel Corporation and its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

Index

CEDT_CFMWS_WINDOW_RESTRICTIONS, [31](#)

CXL_CEDT_HOB, [31](#)

CxlCedtHob.h, [37](#)

CxlNodeHob.h, [37](#)

FSP_ERROR_INFO_HOB, [31](#)

FspAcpiHobs.h, [38](#)

FspEdpcParam.h, [38](#)

FspErrorInfoHob.h, [39](#)

FspExtMemoryPprHob.h, [40](#)

FspExtSystemMemoryMapHob.h, [40](#)

FspGlobals.h, [41](#)

FspiUpd.h, [42](#)

FspmUpd.h, [43](#)

FspUpd.h, [44](#)

FsptUpd.h, [44](#)

FspUpd.h, [45](#)

lioPcieConfigUpd.h, [46](#)

lioUniversalDataHob.h, [47](#)

MEM_SKTCH_TO_IMC

MemoryMapDataHob.h, [48](#)

MemoryMapDataHob.h, [47](#)

MEM_SKTCH_TO_IMC, [48](#)

PLATFORM_DATA, [32](#)

PrevBootErrSrcHob.h, [48](#)

SystemInfoHob.h, [49](#)

UDS_PCIRoot_RES, [33](#)

UDS_SOCKET_RES, [34](#)

UDS_STACK_RES, [34](#)