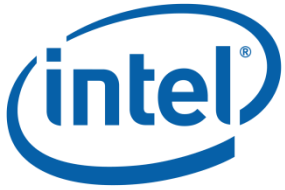


Framework Intel® Processor Reference Code for Haswell

Reference Code Specification

Nov 2014

Version 1.9.0

Intel Confidential

Revision History

Revision Number	Description	Revision Date
0.4.0	Initial version	Nov, 2011
0.5.0	PFAT & PPM Module updated based on HSW BWG 0.50 TXT one touch support changed.	Jan, 2012
0.5.5	Updated to HSW BWG 0.5.5, for Alpha 1 release	Mar, 2012
0.5.6	Updated to HSW BWG 0.5.6, for Post Alpha 1 release	May, 2012
0.6.0	Updated to HSW BWG 0.6.0, for Post Alpha 2 release	June, 2012
0.6.1	Updated to HSW BWG 0.6.1, for ULT Pre Alpha release	Aug, 2012
0.7.0	Updated to HSW BWG 0.7.0 release	Sep, 2012
0.7.1	Updated to HSW BWG 0.7.1 release	Oct, 2012
0.7.2	Updated to HSW BWG 0.7.2 release	Oct, 2012
0.8.0	Updated to HSW BWG 0.8.0 release for HSW 2 Chip Pre-QS	Nov, 2012
0.8.1	Updated to HSW BWG 0.8.1 release for HSW ULT Beta	Dec, 2012
0.9.0	Updated to HSW BWG 0.9.0 release for HSW PC	Jan, 2013
1.0.0	Updated to HSW BWG 1.0.0 release for HSW PV	Jan, 2013
1.1.0	Updated to HSW BWG 1.1.0 release for HSW ULT QS ***Note : CPU RC's Anchor Cove name change to Boot Guard**	Feb, 2013
1.2.0	Updated to HSW BWG 1.2.0 release for HSW ULT PC	Feb, 2013
1.3.0	Updated to HSW BWG 1.3.0 release for HSW ULT PV	Mar, 2013
1.4.0	Updated to HSW BWG 1.4.0 release for CRW PV	Apr, 2013
1.5.0	Updated to HSW BWG 1.5.0 release	May, 2013
1.6.0	Updated to HSW BWG 1.6.0 release	June, 2013
1.6.1	Updated to HSW BWG 1.6.1 release WinBlue 2chip+ULT PV	July, 2013
1.6.2	Updated to HSW BWG 1.6.2 release	Aug, 2013
1.7.0	Updated to HSW BWG 1.7.0 release	Nov, 2013
1.8.0	Updated to HSW BWG 1.8.0 release	Mar, 2014
1.9.0	Updated to HSW BWG 1.9.0 release	Nov, 2014

Contents

1 Introduction

1.1 Purpose and Scope of this Document

The Intel® Processor Reference Code is a set of source code modules that are compliant with the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework") specifications and is used to initialize and control the Intel processor in the system pre-boot process.

This specification describes the high-level design of the Intel® Processor Reference Code and does the following:

- Describes architecture of the Intel Processor Reference Code
- Describes the coding requirements for use of Intel Processor Reference Code
- Describes the system block diagram and basic components of the Intel Processor Reference Code
- Specifies the API -- services, protocols, PPI, functions, and type definitions that are architecturally implemented in the Intel Processor Reference Code
- Describes the guidelines for integrating the reference code to a code base.

This specification is useful for any software developer that utilizes the Intel Processor Reference Code in a Framework compliant BIOS code base.

This document describes the components and their software interfaces in sufficient detail for system BIOS developers to use the Intel Processor Reference Code.

A full understanding of the UEFI and Framework specifications is assumed throughout this document.

The specification is not intended to provide implementation details.

1.2 Related Information

The following publications and sources of information may be useful to you or are referred to by this specification:

Item	Location
Extensible Firmware Interface Specification, Version 1.10, Intel, 2001,	http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-homepage-general-technology.html
Unified Extensible Firmware Interface Specification, Version 2.0, Unified EFI, Inc, 2006	http://www.uefi.org.
Unified Extensible Firmware Interface Specification, Version 2.1, Unified EFI, Inc, 2007	http://www.uefi.org.
Unified Extensible Firmware Interface Specification, Version 2.2, Unified EFI, Inc, 2008	http://www.uefi.org
Intel® Platform Innovation Framework for EFI Specifications, Intel, 2006	http://www.intel.com/technology/framework/download.htm
EDK II Glue Library Programming Guide, Intel, 2007	http://sourceforge.net/projects/efidevkit/files/Documents/Glue_Library.pdf/download
Haswell Processor Family - BIOS Writer's Guide (BWG)	
System Management BIOS (SMBIOS) Specification	http://www.dmtf.org/standards/smbios
RS – Intel Trusted Execution Technology BIOS Specification, Revision 2.1	
Intel® Trusted Execution Technology Preliminary Architecture Specification (or replacement)	http://www.intel.com/technology/security/
Intel Initiatives TPM NV Storage Interface Usage, revision 0.7 or later.	
TPM Main. Specification Version 1.2, Revision 1.09 or later	https://www.trustedcomputinggroup.com
Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 3A: System Programming Guide, Part1.	http://www.intel.com/products/processor/manuals/index.htm
RS - Haswell System Agent BIOS Specification	
Intel® Trusted Execution Technology (TXT) – One-Touch Enabling 0.5	Document no - 481137

1.3 Acronyms and Definitionss

The following terms are used throughout this document to describe varying aspects of input localization:

Item	Description
ACPI	Advanced Configuration and Power Interface.
BDS	Framework Boot Device Selection phase.
Component	An executable image. Components defined in this specification support one of the defined module types.
DPPM	Dynamic Power Performance Module
DXE	Framework Driver Execution Environment phase.
DXE SAL	A special class of DXE module that produces SAL Runtime Services. DXE SAL modules differ from DXE Runtime modules in that the DXE Runtime modules support Virtual mode OS calls at OS runtime and DXE SAL modules support intermixing Virtual or Physical mode OS calls.
DXE SMM	A special class of DXE module that is loaded into the System Management Mode memory.
DXE Runtime	Special class of DXE module that provides Runtime Services
EFI or UEFI	Generic terms that refer to one of the versions of the EFI specification: EFI 1.02, EFI 1.10, UEFI 2.0, or UEFI 2.1.
EFI 1.10 Specification	Intel Corporation published the Extensible Firmware Interface Specification. Intel donated the EFI specification to the Unified EFI Forum, and the UEFI now owns future updates of the EFI specification. See UEFI Specifications.
Foundation	The set of code and interfaces that glue implementations of UEFI together.
Framework	Intel® Platform Innovation Framework for EFI consists of the Foundation, plus other modular components that characterize the portability surface for modular components designed to work on any implementation of the Tiano architecture.
GUID	Globally Unique Identifier. A 128-bit value used to name entities uniquely. An individual without the help of a centralized authority can generate a unique GUID. This allows the generation of names that will never conflict, even among multiple, unrelated parties.
HII	Human Interface Infrastructure. This generally refers to the database that contains string, font, and IFR information along with other pieces that use one of the database components
IFR	Internal Forms Representation. This is the binary encoding that is used for the representation of user interface pages.
iVR	Integrated Voltage Regulator
Library Class	A library class defines the API or interface set for a library. The consumer of the library is coded to the library class definition. Library classes are defined via a library class.h file that is published by a package. See the *EDK 2.0 Module Development Environment Library Specification* for a list of libraries defined in this package.

Item	Description
Library Instance	An implementation of one or more library classes. See the *EDK 2.0 Module Development Environment Library Specification* for a list of library defined in this package.
Module	A module is either an executable image or a library instance. For a list of module types supported by this package, see module type.
Module Type	All libraries and components belong to one of the following module types: BASE, SEC, PEI_CORE, PEIM, DXE_CORE, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SMM_DRIVER, DXE_SAL_DRIVER, UEFI_DRIVER, or UEFI_APPLICATION. These definitions provide a framework that is consistent with a similar set of requirements. A module that is of module type BASE, depends only on headers and libraries provided in the MDE, while a module that is of module type DXE_DRIVER depends on common DXE components. For a definition of the various module types, see module type.
OC	Over Clocking
Package	A package is a container. It can hold a collection of files for any given set of modules. Packages may be described as one of the following types of modules: • source modules, containing all source files and descriptions of a module • binary modules, containing UEFI Sections or a Framework File System and a description file specific to linking and binary editing of features and attributes specified in a Platform Configuration Database (PCD,) • mixed modules, with both binary and source modules Multiple modules can be combined into a package, and multiple packages can be combined into a single package.
Protocol	An API named by a GUID as defined by the UEFI specification.
PCD	Platform Configuration Database.
PEI	Pre-EFI Initialization Phase.
PPI	A PEIM-to-PEIM Interface that is named by a GUID as defined by the PEI CIS.
SAL	System Abstraction Layer. A firmware interface specification used on Intel® Itanium® Processor- based systems.
Runtime Services	Interfaces that provide access to underlying platform-specific hardware that might be useful during OS runtime, such as time and date services. These services become active during the boot process but also persist after the OS loader terminates boot services.
SCI	System Control Interrupt
SEC	Security Phase is the code in the Framework that contains the processor reset vector and launches PEI. This phase is separate from PEI because some security schemes require ownership of the reset vector.
SSDT	Secondary System Definition Table

Item	Description
UEFI Application	An application that follows the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.
UEFI Driver	A driver that follows the UEFI specification.
UEFI Specification Version 2.1	Current version of the UEFI specification released by the Unified EFI Forum. This specification builds on the EFI 1.10 specification and transfers ownership of the EFI specification from Intel to a non-profit, industry trade organization.
Unified EFI Forum	A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see www.uefi.org.

1.4 Processor Terms

Item	Description
AP	Application Processor. All the processor threads other than SBSP
Cache Line	Is defined as being 64-bytes in size, with each line having a unique set of MESI bits.
CSI	Intel® QuickPath Interconnect (Intel® QPI) - formerly known as Common System Interface (CSI). A cache-coherent, link-based interconnect specification for Intel processor, chipset, and I/O bridge components.
CSR	Configuration Space Register or Control and Status register. PCI/PCIe family style registers used to define configurable resources and features.
CMP	Core Multi-Processing (CMP) refers to a single physical package that utilizes multiple cores for multi-processing capabilities.
Core	The silicon which contains 1 or more logical processors (with or without Intel® Hyper-Threading Technology).
CPU-only RESET	Any assertion of the processor RESET# input signal that does not also assert the PCI RESET signal. The PWRGOOD# signal is not toggled. This capability is not supported by all chipsets.
Enhanced Thermal Control Circuit	Similar to the TCC, the Enhanced Thermal Control Circuit (Enhanced TCC) is a feature of the processor that is used to cool the processor should the processor temperature exceed a predetermined activation temperature. The Enhanced TCC reduces the processor core to bus ratio, and VID when active.
HARD RESET	Assertion of the RESET# input signal of the processor and also asserts the PCI RESET signal. The PWRGOOD# signal is not toggled.
Integrated Memory Controller (IMC)	A memory controller that is integrated in the processor silicon.

Item	Description
Intel® TM	Intel® Thermal Monitor (Intel® TM) is a feature of the processor. The thermal monitor contains TCC. When Intel TM is enabled and active due to the die temperature reaching the pre-determined thermal monitor activation temperature, the TCC attempts to cool the processor by alternating stopping the processor clocks for a period of time, then allows them to run full speed for a period of time (duty cycle ~30% – 50%) until the processor temperature drops below the activation temperature.
Intel® TM2	Intel® Thermal Monitor 2 (Intel® TM2) is a feature of the processor. The Intel Thermal Monitor 2 contains the Enhanced TCC. When Intel TM2 is enabled and active due to the die temperature reaching the predetermined Intel Thermal Monitor activation temperature, the Enhanced TCC attempts to cool the processor by first reducing the core to a specific bus ratio, then steps down the VID.
IPI	An "Inter-Processor Interrupt" is a message from one processor to another processor. The IPI is sent by the local XAPIC when software writes to the command register in the local XAPIC. The IPI is sent using the system bus.
Logical Processor	The basic unit of processor hardware that allows the software executive in the operating system to dispatch a task or execute a thread context. Each logical processor can execute only one thread context at a time. A processor that may share execution resources with other processors in the same core.
Multi-Core Processor	A physical package that contains more than one processor core.
MSR	Model Specific Register (MSR) as the name implies is model specific and may change from processor model number (n) to processor model number (n+1). An MSR is accessed by setting ECX to the register number and executing either the RDMSR or WRMSR instruction. The RDMSR instruction will place the 64 bits of the MSR in the EDX:EAX register pair. The WRMSR writes the contents of the EDX:EAX register pair into the MSR.
NBSP or Node BSP	Hardware within each processor package selects one logical processor to be the Bootstrap Processor. This is called Node BSP. There is no guarantee that the HW selected NBSP will have an intra-processor APIC ID == 00h. It is also possible that the package does not contain an enabled core with an intra-processor APIC ID == 00h.
Physical Processor	A package which contains 1 or more cores that share a common connection to the system bus.
System BSP or SBSP	The single Node BSP selected by BIOS from all processor packages installed in the platform, to perform most of the BIOS initialization steps and to hand off control to the OS. In a DP system, one processor package contains the SBSP and the other processor package contains a NBSP. In a UP system, the one processor package contains the SBSP and no NBSP exists. There is no guarantee that the SBSP will have an intra-processor APIC ID == 00h. It is also possible that the package(s) does not contain an enabled core with an intra-processor APIC ID == 00h.
Sector	Is defined as 2 cache-lines. An even and odd cache line pair comprises a sector. Not all caches are sectored.
System Bus	"System Bus" refers to the interface between the processor, system core logic (the chipset components), and other bus agents. On previous processor generations this was implemented as FSB.
System RESET	Same as "HARD RESET".
TCC	The Thermal Control Circuit (TCC) is a feature of the processor that is used to cool the processor should the processor temperature exceed a predetermined temperature.
Thread	A Logical Processor

Item	Description
System Agent	Refers to the functionality in processor package other than processor cores. The System Agent encompasses the System Bus interface and support logic, integrated memory controller, shared cache, etc.
VID	Voltage Identification (VID) is a binary pattern output from the processor that tells the voltage regulator the voltage required to operate the processor.
WARM RESET	Assertion of the INIT# input signal of the processor. Cache coherency is maintained. Does not include "CPU-only RESET", or "HARD RESET".
PFAT	Platform Firmware Armoring Technology
PUPC	PFAT Update Package Certificate
PUP	PFAT Update Package
PPDT	PFAT Platform Data Table
TxT	Trusted Execute Technology
Boot Guard	Boot Guard (formerly known as Anchor Cove)
ACM	Authenticated Code Module

1.5 Conventions Used in This Document

This document uses the typographic and illustrative conventions described below.

1.5.1 Data Structure Descriptions

Intel® processors based on 32 bit Intel® architecture (IA 32) are "little endian" machines. This distinction means that the low-order byte of a multi-byte data item in memory is at the lowest address, while the high-order byte is at the highest address. Processors of the Intel® Itanium® processor family may be configured for both "little endian" and "big endian" operation. All implementations designed to conform to this specification will use "little endian" operation.

In some memory layout descriptions, certain fields are marked reserved. Software must initialize such fields to zero and ignore them when read. On an update operation, software must preserve any reserved field.

The data structures described in this document generally have the following format:

STRUCTURE NAME	The formal name of the data structure.
Summary	A brief description of the data structure.
Prototype	A "C-style" type declaration for the data structure.
Parameters	A brief description of each field in the data structure prototype.
Description	A description of the functionality provided by the data structure, including any limitations and caveats of which the caller should be aware.

Related Definitions	The type declarations and constants that are used only by this data structure

1.5.2 Protocol Descriptions

The protocols described in this document generally have the following format:

Protocol Name	The formal name of the protocol interface.
Summary	A brief description of the protocol interface.
GUID	The 128-bit Globally Unique Identifier (GUID) for the protocol interface.
Protocol Interface Structure	A "C-style" data structure definition containing the procedures and data fields produced by this protocol interface.
Parameters	A brief description of each field in the protocol interface structure.
Description	A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.
Related Definitions	The type declarations and constants that are used in the protocol interface structure or any of its procedures.

1.5.3 Procedure Descriptions

The procedures described in this document generally have the following format:

ProcedureName()	The formal name of the procedure.
Summary	A brief description of the procedure.
Prototype	A "C-style" procedure header defining the calling sequence.
Parameters	A brief description of each field in the procedure prototype.
Description	A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.
Related Definitions	The type declarations and constants that are used only by this procedure.
Status Codes Returned:	A description of any codes returned by the interface. The procedure is required to implement any status codes listed in this table. Additional error codes may be returned, but they will not be tested by standard compliance tests, and any software that uses the procedure cannot depend on any of the extended error codes that an implementation may provide.

1.5.4 Instruction Descriptions

The procedures described in this document generally have the following format:

InstructionName	The formal name of the instruction.
SYNTAX	A brief description of the instruction.
DESCRIPTION	A description of the functionality provided by the instruction accompanied by a table that details the instruction encoding.

OPERATION	Details the operations performed on operands.
BEHAVIORS AND RESTRICTIONS	An item-by-item description of the behavior of each operand involved in the instruction and any restrictions that apply to the operands or the instruction.

1.5.5 PPI Descriptions

A PEIM-to-PEIM Interface (PPI) description generally has the following format:

PPI Name	The formal name of the PPI.
Summary	A brief description of the PPI.
GUID	The 128-bit Globally Unique Identifier (GUID) for the PPI.
Protocol Interface Structure	A "C-style" procedure template defining the PPI calling structure.
Parameters	A brief description of each field in the PPI structure.
Description	A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.
Related Definitions	The type declarations and constants that are used only by this interface.
Status Codes Returned	A description of any codes returned by the interface. The PPI is required to implement any status codes listed in this table. Additional error codes may be returned, but they will not be tested by standard compliance tests, and any software that uses the procedure cannot depend on any of the extended error codes that an implementation may provide.

1.5.6 Pseudo-Code Conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a list is an unordered collection of homogeneous objects. A queue is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be First In First Out (FIFO).

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the Extensible Firmware Interface Specification.

1.5.7 Typographic Conventions

This document uses the typographic and illustrative conventions described below:

Plain text (Body)	The normal text typeface is used for the vast majority of the descriptive text in a specification.
Plain text [blue] (Cross-Reference)	Any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink. USE ONLY IF YOU MAKE AN ACTUAL CROSS-REFERENCE LINK.
Bold (Bold or GlossTerm)	In text, a Bold typeface identifies a processor register name. In other instances, a Bold typeface can be used as a running head within a paragraph. or as a definition heading (GlossTerm)
Italic	In text, an Italic typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.
BOLD Monospace (CodeCharacter and CodePargraph)	Computer code, example code segments, and all prototype code segments use a **BOLD Monospace** typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.
Bold Monospace	Words in a **Bold Monospace** typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink. USE ONLY IF YOU MAKE AN ACTUAL HYPERLINK.
Italic Monospace *(ArgCharacter and ArgParagraph)*	In code or in text, words in Italic Monospace indicate placeholder names for variable information that must be supplied (i.e., arguments).
Plain Monospace (CodeCharacter + Not Bold)	In code, words in a Plain Monospace typeface that is a dark red color but is not bold or italicized indicate pseudo code or example code. These code segments typically occur in one or more separate paragraphs.
text text text	In the PDF of this specification, text that is highlighted in yellow indicates that a change was made to that text since the previous revision of the PDF. The highlighting indicates only that a change was made since the previous version; it does not specify what changed. If text was deleted and thus cannot be highlighted, a note in red and highlighted in yellow (that looks like *(Note: text text text.)*) appears where the deletion occurred.

See the glossary sections in the *EFI 1.10 Specification* and in the *EFI Documentation* help system for definitions of terms and abbreviations that are used in this document or that might be useful in understanding the descriptions presented in this document.

See the references sections in the *EFI 1.10 Specification* and in the in the *EFI Documentation* help system for a complete list of the additional documents and

specifications that are required or suggested for interpreting the information presented in this document:

The *EFI 1.10 Specification* is available from the EFI web site http://developer.intel.com/technology/efi/. The *EFI Documentation* help system is available from the EFI web site http://developer.intel.com/technology/efi/help/efidocs.htm.

See the master Framework glossary in the *Framework Interoperability and Component Specifications* help system for definitions of terms and abbreviations that are used in this document or that might be useful in understanding the descriptions presented in this document.

See the master Framework references in the *Interoperability and Component Specifications* help system for a complete list of the additional documents and specifications that are required or suggested for interpreting the information presented in this document.

The Framework Interoperability and Component Specifications help system is available at the following URL:

http://www.intel.com/technology/framework/spec.htm

§

2 Overview

2.1 Architectural Overview

This reference code provides the Intel processor support for the Haswell Client processors. It is organized into separate modules and generally categorized as follows:

Intel Processor Support

- DXE Driver – Features Init, Microcode Update and MP Services init.
- PEIM – Provide Cache PPI and build BistHob from SecPlatformInformationPpi from SEC, and CPU Strap support.
- S3 PEIM – Provide MP initialization on the S3 resume path.
- PowerManagement – Provide DXE driver to initialize the Processor Power Management, and SMM driver to saves and restores the Power Management related MSRs during S3. It also provides ACPI tables for C-state, P-state, T-state.
- DTS – Provide SMM driver for on-die CPU thermal reporting.
- TXT – Provide PEI and DXE for TXT environment initialization and module initialization for TPM and ACM.
- PFAT - Provide flash protection and BIOS update authorization using a combination of CPU and PCH features.

Recommendations and requirements for code modifications in SEC driver and platform specific modules are also described in these below sections:

- Publish protocol
- SEC Sample Code - Processor Init Sample Code for different SEC build design
- Platform Consideration – The most important part for platform code is to provide the MTRR layout in PEIM when memory is ready

2.2 Rationale

This section describes the basic rationale behind providing Framework compliant Intel Processor Reference Code:

- Provide Intel processor support sufficient for the Intel processor BWG requirement
- Provide code easily portable to multiple platforms
- Provide code to easily support multiple platforms without modifying the Intel processor modules

2.3 Requirements

This section captures the requirements of Intel Processor Reference Code:

- Framework compliant
  - Build in EDK version 20061117, AKA EDK Release 1.01 with Patch #8
  - Compliant to the UEFI 2.1 specification
  - Compliant to the Framework 0.9x specifications
    - Except for the HII specification which is superseded by the UEFI 2.1 HII
- Use the interfaces on EDKII Glue Library
- No use of platform or other silicon specific code or definitions
- Easily portable to different Framework compliant code bases utilizing the EDK
- Easily portable to EDKII with minimal source code modifications
- Easily portable to different platforms
- Implement Intel Haswell BIOS Writer's Guide requirements and recommendations

§

3 Intel® Processor

Processor initialization requirements are generally documented in the Intel® 64 and IA-32 Architectures Software Developer's Manuals and the BIOS Writer's Guide(s) corresponding to the processor(s) supported.

3.1 Processor Initialization

3.1.1 Processor Initialization Flow

There high-level flow for processor initialization is:

- Early initialization near reset vector to set protected mode, load microcode updates, initialize NEM mode.
- CPU PEI performs early initialization related to configuration prior to MemoryInit.
- Initialize memory. This is covered in greater detail in the System Agent reference code and Memory Reference Code documentation
- Transfer early cache data contents into memory, transition to a standard caching configuration
- Platform code to provide the cache layout to configure cache.
- Fully configure processor, including MP initialization, final cache configuration
- Fully configure processor power management. This is covered in greater detail in the Power Management section 3.2
- Configure Intel® TXT. This is covered in greater detail in the TXT section 3.4
- Put processors in OS handoff state

OS will typically reinitialize some processor configuration, most specifically, OS will typically reconfigure cache and reinitialize MP configuration.

On S3:

- Some MSR state will be saved
- System enters S3
- System receives a wake event
- BIOS follows a similar early initialization flow
- Cache configuration is restored
- MP configuration is restored
- SMM configuration is restored

3.1.1.1 OS handoff state is restored SEC Initialization

- Only Sample Code is provided in processor reference code package for SEC
- SEC enters flat 32 bit protected mode
- SEC loads BSP MCU and enables CSR access if microcode is fail to load by FIT pointer.
- SEC BSP MCU must be loaded before CSR access
- SEC initializes No Eviction Mode
- SEC establishes the stack for PEI use
- BIST data is stored in EFI_SEC_PLATFORM_INFORMATION_PPI to be used by processor PEIM for creating EFI_HT_BIST_HOB_GUID for processor Init DXE driver to report the health status via MP Services

Please refer to Haswell BIOS Writer's Guide for details related to the SEC sample code.

3.1.1.2 PEI Initialization

- Performs early initialization related to configuration
- Install PEI_CACHE_PPI for platform MTRR configuration services
- Install notification for PCH_INIT_PPI
    - Upon notification, configure strap options
- Initializes Performance and Power management features
- Enable XMM
- PFAT Initialization

3.1.1.3 DXE Initialization

- Perform MP initialization per the Software Developer's Manual
- **Install Exception Handler** - CPU Exception Handler is provided by Intel® Processor Reference Code. Usually people can see the hang at Processor code because of exception. With the debug build, you can see more exception detail information from debug serial out.
- **Install EFI_CPU_ARCH_PROTOCOL** – CPU Architectural Protocol to provide the interface to abstract processor-specific functions from the DXE Foundation. The interfaces include flushing caches, enabling, disabling interrupts, hooking interrupt vectors and exception vectors, reading internal processor timers, resetting the processor, and determining the processor frequency. Please refer Driver Execution Environment Core Interface Specification (DXE CIS)** v0.91 for detail information.
- **Sync GCD with MTRRs configuration** – This sync is to update GCD Memory Space Attributes upon MTRRs settings.
- **Locate CPU Platform Policy Protocol** – All setup configurations are provided by CPU Platform Policy Protocol which is installed by platform code after checking CPU Setup options. .

- **Get Software SMI number from Platform Policy for SMMBASE Init** – CPU Init DXE driver has to use SMI to copy data to SMRAM due to SMRR. We need platform code to provide a software SMI number to do some basic init in SMI from DXE drivers.

- **Locate EFI_DATA_HUB_PROTOCOL –** CPU DXE Init driver will use this protocol to log CPU Information into DataHub. So that SMBIOS driver can convert information in DataHub to SMBIOS type 4 and 7 for Processor and Cache info reporting. Please refer Data Hub Specification for more detail information.

- **Load Microcode to memory for POST and copy Microcode to SMM Memory for S3 to perform Microcode update** – During S3 resume, BIOS will get Microcode code from SMM Memory which is prepared by SMM_FROM_CPU_DRIVER_SAVE_INFO service of SMMBASE Software SMI.

- **Prepare HII data for basic string to be logged in DataHub** – There are some basic HII data, such as Intel® Genuine name Processor, Intel® corporation, and unknown.

- **Init Multi-Processors**

    - InitializeMpSupport is the entry of MP init.

    - MP_CPU_RESERVED_DATA will keep all MP information.

    - **InitializeMpSystemData tasks detail:**

        - Allocate temporary memory for AP to run during POST.

        - Perform INIT-SIPI-SIPI to startup APs

        - **Program basic processor functions** – Program all functions in EFI_MSR_IA32_MISC_ENABLE . Init Thermal Monitor, program MISC functions, and init Machine Check Registers

        - **Collect and program Processor Features** – Features such as XD, VMX, DCA, AES, and XAPIC which need to check CPUID or MSR to continue the enabling.

- **Update Platform CPU Data** – Install CpuInfoProtocolGuid Protocol to report the features enabling result. Platform code, such as Setup, can use this protocol to check the enabling result and show it on Setup Page. This protocol also indicates that CpuDxeInit driver has done the CPU Features programming.
- **Init Processor and Cache DataHub** – log all processor and cache data to DataHub for SMBIOS creation, for example.

- **EfiCreateProtocolNotifyEvent to EXIT_PM_AUTH_PROTOCOL** – Notify code to do some final init, such as reload Microcode relocate memory for APs running before booting to OS.
  - With CSM case, we will use the allocated memory in EBDA for APs running.
  - Without CSM, we will allocate a reserved memory below 640KB for APs running.
- **EfiCreateProtocolNotifyEvent to EFI_LEGACY_BIOS_PROTOCOL** – Notify code to allocate memory in EBDA for APs running when CSM is supposed. The purpose we pull-in the relocation is to allocate memory in EBDA as early as possible without impact different CSM implementation about EBDA memory maintain method.
- **CreateEvent to EVENT_SIGNAL_EXIT_BOOT_SERVICES** – call ResetAps to reset buffer for S3 usage.
- **Publish MP Services Protocol** – provide MP Services to run function for all APs.

3.1.1.4 S3 Initialization

- CpuS3Peim will perform basic CPU restore for CPUs.
  - Retrieve data from SMM memory to reserved memory.
    - Open SMM region

- Locate gSmramCpuDataHeaderGuid in SMM Memory – Assume the last range of SMM memory reported by SmmAccessPpi->GetCapabilities is TSEG. Search gSmramCpuDataHeaderGuid and return the location.
- Copy data structure from SMM memory to reserved memory – Such as AcpiCpuData, S3BootScriptTable, S3BspMtrrTable, and Microcode.
- Close SMM region

- Program BSP virtual wire mode, and Microcode.
- Restore BSP feature setting.
- Restore APs configuration.
- Restore BSP MTRRs configuration.
- Restore APs MTRRs configuration.

3.1.1.5 Microcode Update

3.1.1.5.1 Requirements

Before utilize processor, we need to perform Microcode Update first. For detail information, please refer to Haswell BWG.

3.1.1.5.2 Design

Current implement is to load BSP Microcode in BSP and other APs in DXE phase. For performance considerations, the code reads Microcode from flash one time only to memory and make a copy to TSEG by SMM drivers. During S3 resume, the code will open SMM region and copy the Microcode to reserved memory to perform the Microcode update.

3.1.1.6 Other Considerations

Initializing AP in PEI – Intel® Processor Reference Code doesn't provide MpServices in PEI phase. The exception is TXT. TXT needs to ask APs to run some init code, so TXT will load Microcode Code for APs before running the init.

Flexibility for early silicon – it is fairly common for pre-production silicon to have special requirements on when and where to load microcode updates. So far Intel® Processor Reference Code doesn't provide the example yet.

3.1.1.7 MTRR Configuration

3.1.1.7.1 Requirements

BIOS needs to configure cache for system memory, PCI, etc. BIOS no longer needs to reserve 2 MTRR for OS use. Platform code needs to provide the layout and Intel® Processor Reference Code only provides CachePpi to setup specific range to be specific cache type.

3.1.1.7.2 Design

MTRRs are a limited resource. Below is an example to provide a basic layout in platform code. Please refer to Chapter 3.6.3 for the detail sample code.

1. ResetCache
2. Look into the HOB and check all EFI_HOB_TYPE_RESOURCE_DESCRIPTOR and calculate how much memory is below and above 4G.
3. For below 4G memory
    a、 Set WB flow first
    b、 Set WB/UC flow for reminding memory.
4. For above 4G
    a、 Set WB flow first
    b、 Set WB/UC flow for reminding memory

5. If only reminding one MTRR, there are two options,

   a、 Option 1: enlarge the last WB region. This last MTRR setting will include some non-memory region.

   b、 Option 2: Set the last WB region directly, and leave. The last memory region is in un-cached state. System might slow down when using that region.

6. For 0-640KB -- SetCache to WB

7. For Flash area -- SetCache to WB

3.1.1.8 Other Considerations

Before MRC, Cache will be programmed as memory so BIOS can load PEIM. Once memory is ready after MRC running, platform code needs to provide cache layout for detail MTRR configuration to improve the performance as soon as possible. Besides, MTRR won't cover TSEG. Only SMRR maps to TSEG to improve the performance of SMM code.

Key places to review cache configuration.

- End of SEC – Cache configuration for Execute in Place (XIP) performance
- After Memory Initialization – Cache config for late PEI/DXE
- PEI->DXE handoff – Should be the same as previous
- OS handoff – verify memory map and cache configuration alignment
- SMM SMRR and malicious meddling with MTRR.

3.1.1.9 CPU PEI to DXE handoff information

3.1.1.9.1 Requirements

Platform code need to create CPU hand-off information. GCD uses this information to initialize the GCD services.

3.1.1.9.2 Design

When memory is ready for use, platform code checks the EFI_CPUID_EXTENDED_FUNCTION to identify the physical address size. For Haswell Processor, platform code can use following code to build the handoff information,

```
BuildCpuHob (36, 16);
```

3.1.1.10 MP Services

3.1.1.10.1 Requirements

AP waiting state should be biased towards low latency, but power aware.

3.1.1.10.2 Design

In current implementation, AP waiting state is Monitor/MWAIT loop.

3.1.1.10.3 Other consideration

Polling is functional, but power inefficient.

Wait-For-SIPI is good for power, but high latency and can't response to SMI.

HALT/loop ok, but high latency and high power consumption.

Monitor/MWAIT Cx (depending on how deep current processor supports) is the best.

3.1.1.11 OS Handoff State

3.1.1.11.1 Requirements

- Make sure Page Table is in reserved memory before handing off to OS.
    - If page table is not in reserved memory, OS might destroy page table unconditionally during the boot or S4 resume and APs will not be able to stay in MWAIT state correctly, and result in unexpected behavior, such as reset or hang.
- AP OS handoff state should be lowest power state

- Wait-For-SIPI is ok, but it APs can't response to SMI.
- C1/Halt is ok, but it might consume more power.
- Deepest Cx is best. We have MWAIT support right now.

- AP resume vector should be in memory isolated from OS
  - EBDA – this is current implementation. After booting to OS, OS will allocate different memory for APs to run. This location is also for S3 resume right before BIOS handoff to OS resume vector.
  - 0xE000-0xF000 – The impact to use this method is we will need to add chipset code to unlock EF segment in CpuS3Peim driver.

3.1.1.11.2 Design

In current implementation, during POST, all APs are in MWAIT C1 state, and CPU will be in Halt/Loop State before handing-off to OS in S3 resume path.

- Ready To Boot Event state – MWAIT deepest Cx state
- ExitBootServices state –MWAIT deepest Cx State
- INT19 state –MWAIT deepest Cx State
- S3 Resume OS handoff state – Halt/Loop State

3.1.1.12 S3 SMM Memory Map for Multi-Processors

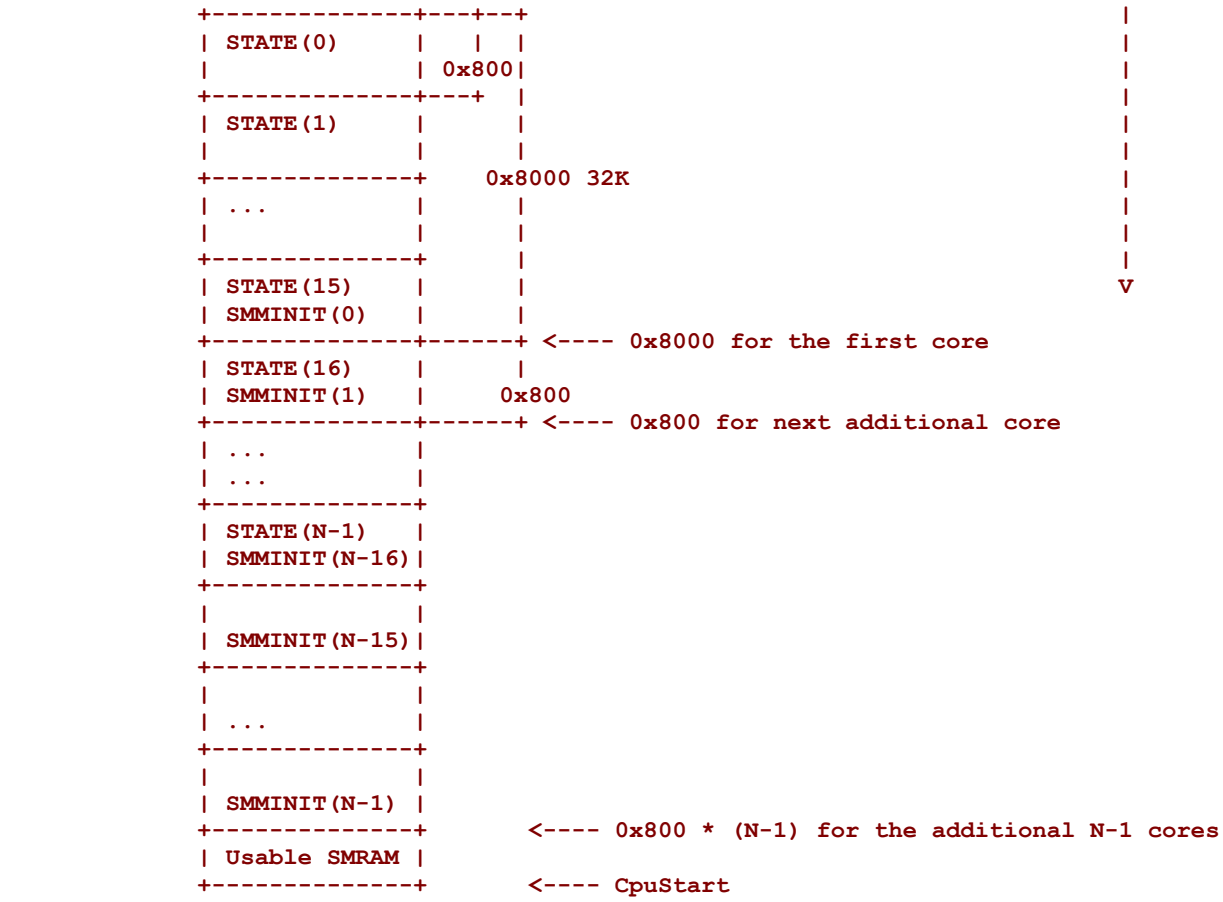
3.1.1.12.1 Requirements

Security considerations – all data should be from SMM Memory. Only SMM Memory is trustable and secured.

3.1.1.12.2 Design

Below is SMRAM Memory Map for TSEG:

```
+--------------+ <- Top of TSEG
| MSEG (Opt)2MB|
+--------------+
| IED (Opt) 2MB|
```



Some key points:

1. Each Processor Save State uses 2K.
2. Processor SMM Entry will be at SMM Base + 0x8000.
3. First processor SMM entry is at SMMINIT(0), at end of SMM Memory – 0x8000.
4. First Processor SMM Save State is at STATE(0) which is at end of SMM memory – 2K.
5. The next SMM Entry will be SMMINIT(0) – 0x800(2k).
6. The next SMM Save State is at STATE(0) – 0x800(2k).
7. Eventually, SMMINIT(0) and STATE(15) will be in 0x800(2k) region.
8. **So SMMINIT code size + CPU Save State has to be less or equal to 0x800(2k) size.**

3.1.1.12.3 Other Consideration

After CPU Remap is done and SMRR is enabled, it is not allowed to update SMM region in normal mode, otherwise when SMI happen, SMRR will be out of sync and caught unexpected result. If you want to copy data into SMM region, you can use software SMI for this purpose.

3.1.2 Modules

3.1.2.1 CPU Init DXE Module

3.1.2.1.1 Introduction

All processor init such as microcode update, MSR init, MP Service, will be done in processor DXE module.

3.1.2.1.2 Prerequisites

- EFI_METRONOME_ARCH_PROTOCOL_GUID
- EFI_CPU_IO_PROTOCOL_GUID
- EFI_HII_PROTOCOL_GUID for UEFI2.0
- EFI_HII_DATABASE_PROTOCOL_GUID for UEFI2.1 or above
- EFI_DATA_HUB_PROTOCOL_GUID
- EFI_VARIABLE_ARCH_PROTOCOL_GUID
- EFI_VARIABLE_WRITE_ARCH_PROTOCOL_GUID
- DXE_CPU_PLATFORM_POLICY_PROTOCOL_GUID

3.1.2.1.3 Prerequisite

processor DXE will produce EFI_MP_SERVICES_PROTOCOL.

3.1.2.1.4 Integration Check List

None

3.1.2.1.5 Porting Recommendation

None

3.1.2.2 CPU Init PEIM

3.1.2.2.1 Introduction

CPU PEIM provides PEI_CACHE_PPI and builds EFI_HT_BIST_HOB_GUID.

3.1.2.2.2 Prerequisites

- PEI_CPU_PLATFORM_POLICY_PPI_GUID

3.1.2.2.3 Integration Check List

- Make sure EFI_HT_BIST_HOB_GUID is built.
- CPU Init PEIM needs gPchInitPpiGuid to set processor strap and perform reset when needed.

3.1.2.2.4 Porting Recommendation

None

3.1.2.3 CPU S3 PEIM

3.1.2.3.1 Introduction

The PEI module to initialize the Processor during S3 resume.

3.1.2.3.2 Prerequisites

- PEI_MASTER_BOOT_MODE_PEIM_PPI
- PEI_ PERMANENT_MEMORY_INSTALLED_PPI_GUID
- PEI_SMM_ACCESS_PPI_GUID

3.1.2.3.3 Integration Check List

None

3.1.2.3.4 Porting Recommendation

None

3.1.2.4 SMBIOS Table Integration

3.1.2.4.1 Introduction

SMBIOS code is included in CPU DXE Driver. The code will build Processor/Cache Datahub for SMBIOS Type 4/7 creation.

3.1.2.4.2 Prerequisite

- **`EFI_DATA_HUB_PROTOCOL`** – Documented in *Data Hub Specification*, available at the URL: http://www.intel.com/technology/framework/spec.htm
- **`EFI_HII_PROTOCOL`** – Documented in *Human Interface Infrastructure Specification*, available at the URL: http://www.intel.com/technology/framework/spec.htm

3.1.2.4.3 Result

Publishes SMBIOS data structures related to the System Agent. Refer to the Data Hub Memory Subclass Specification, available at the URL: http://www.intel.com/technology/framework/spec.htm

3.1.2.4.4 Validation Check List

Check the boot result and check if SMBIOS type 4/7 is there.

3.2 Power Management

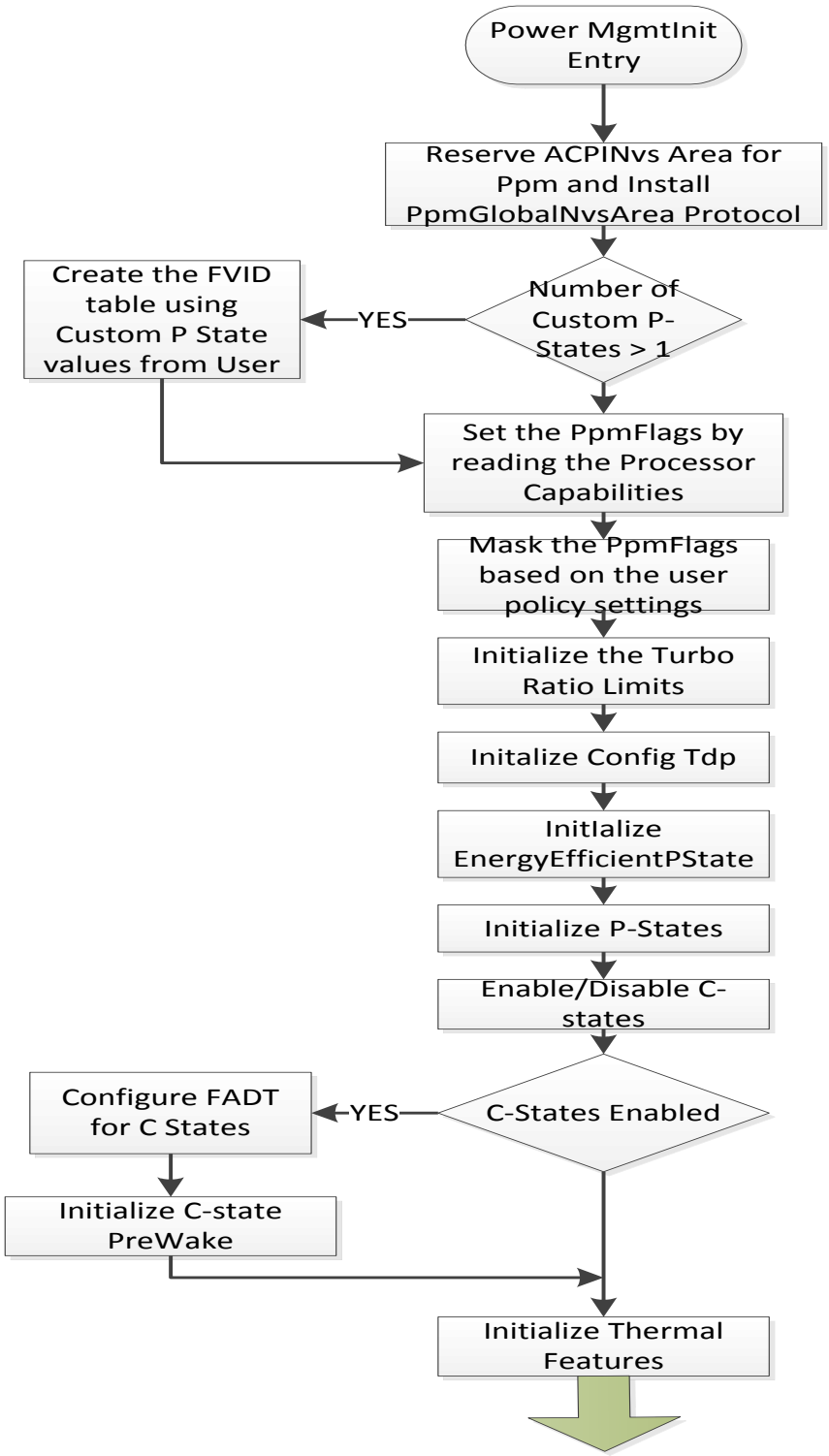
Power Management Module is a Framework compliant Dxe and SMM driver with ACPI 5.0 support. Power Management Module Provide full support for Haswell processor family power management features as specified in the Haswell *Processor family BIOS Writer's Guide*.

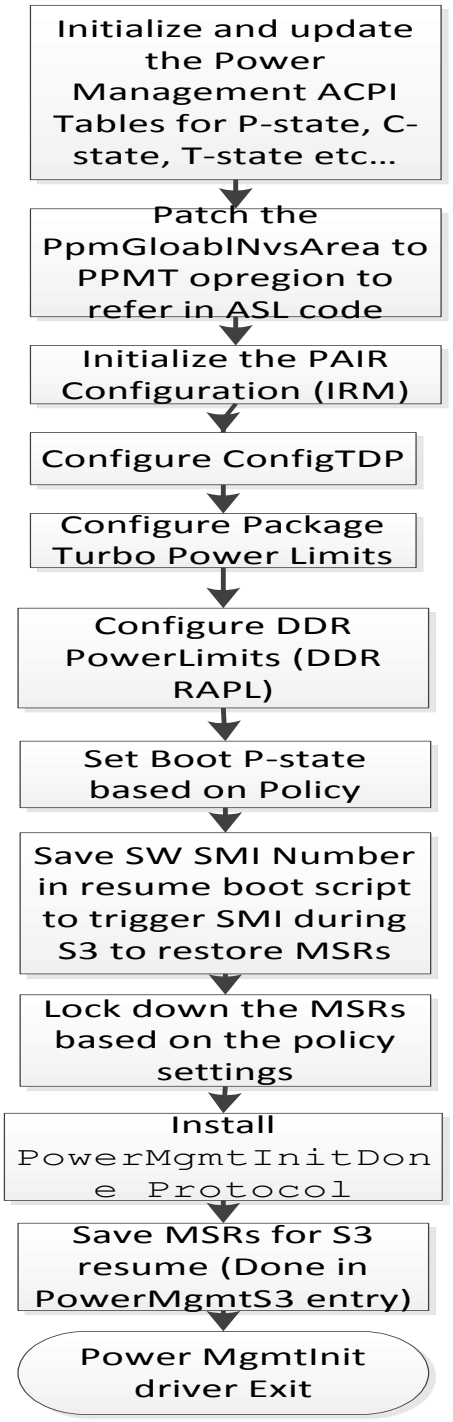
Centralize all Processor power management code in a single Dxe driver

- Support single and dual and quad core processor configurations
- Support Core P-States
- Support Core C-States
- Support Hardware/Software coordinated P-States
- Support Monitor/MWAIT style C state entry
- Support I/O MWAIT redirection style C state entry
- Support custom P-states
- Support C-state pre-wake
- Configure Turbo Ratio limits
- Configure EnergyEfficient P state
- Configure PAIR (IRM) configuration
- Support Intel® Thermal features
- Configure ConfigTDP
- Support IA Package and DDR Turbo power limits (RAPL)
- Dynamically load the C-state and P-state ACPI tables based on configuration and OS driver capabilities

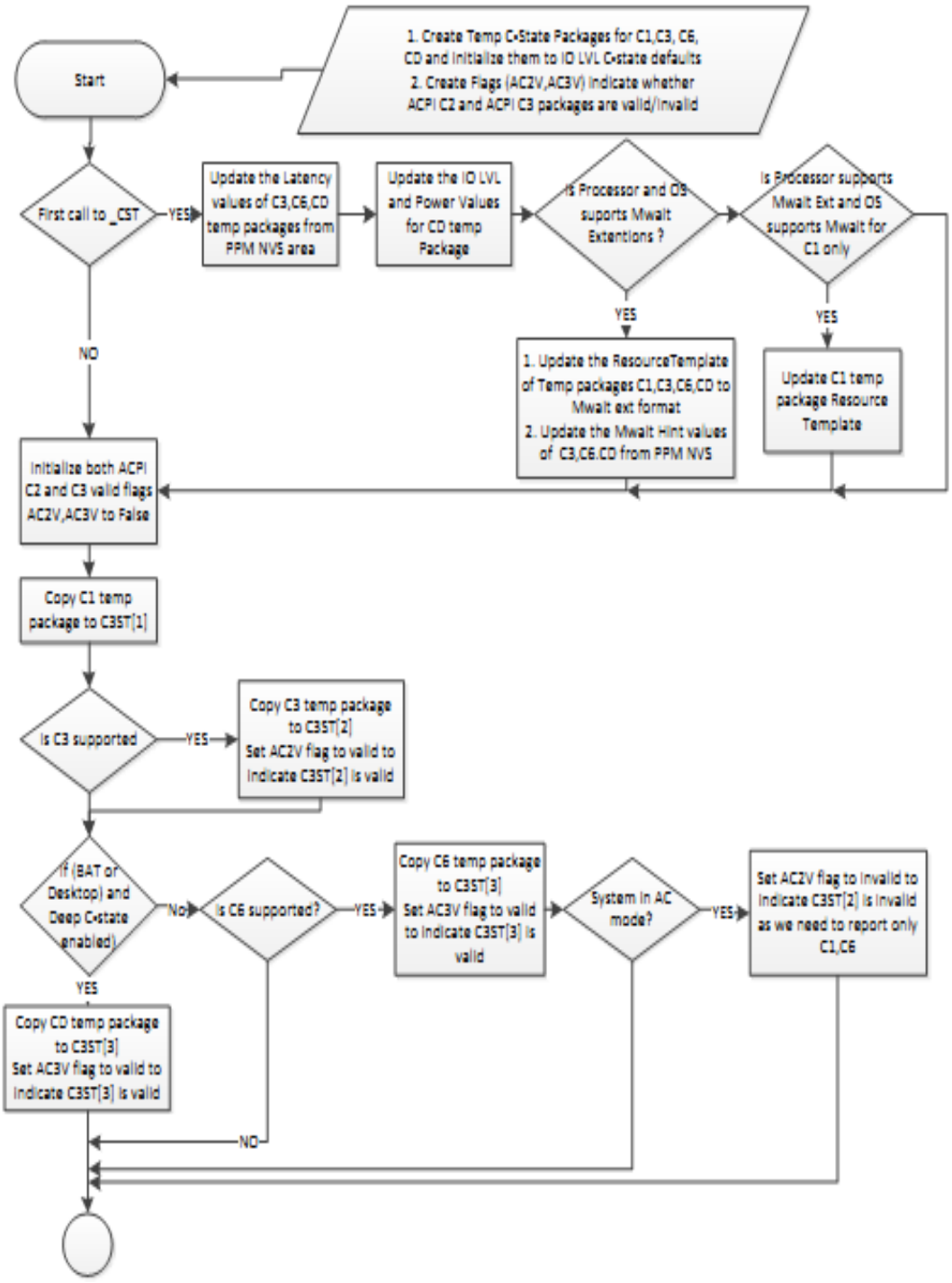
Power Management Initialization is done after BIOS_RESET_CPL configuration is done. System Agent Reference Code sets BIOS_RESET_CPL in System Agent PEIM.

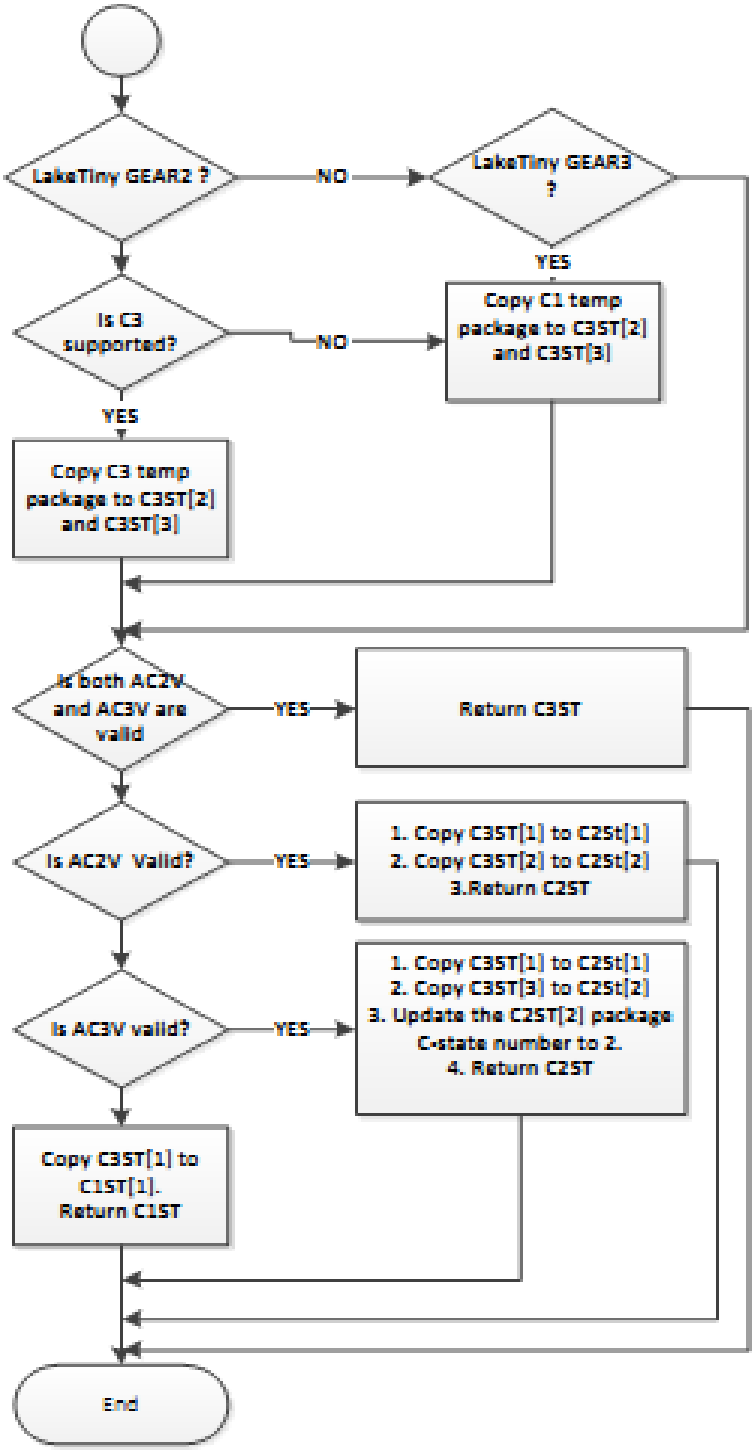
3.2.1 Power Management Execution Flow


Power MgmtInit Entry
Reserve ACPINvs Area for Ppm and Install PpmGlobalNvsArea Protocol
Number of Custom P-States > 1
YES
Create the FVID table using Custom P State values from User
Set the PpmFlags by reading the Processor Capabilities
Mask the PpmFlags based on the user policy settings
Initialize the Turbo Ratio Limits
Initalize Config Tdp
Initlalize EnergyEfficientPState
Initialize P-States
Enable/Disable C-states
C-States Enabled
YES
Configure FADT for C States
Initialize C-state PreWake
Initialize Thermal Features


Initialize and update the Power Management ACPI Tables for P-state, C-state, T-state etc...
Patch the PpmGloablNvsArea to PPMT opregion to refer in ASL code
Initialize the PAIR Configuration (IRM)
Configure ConfigTDP
Configure Package Turbo Power Limits
Configure DDR PowerLimits (DDR RAPL)
Set Boot P-state based on Policy
Save SW SMI Number in resume boot script to trigger SMI during S3 to restore MSRs
Lock down the MSRs based on the policy settings
Install PowerMgmtInitDone Protocol
Save MSRs for S3 resume (Done in PowerMgmtS3 entry)
Power MgmtInit driver Exit

3.2.2 _CST Flow


Start
1. Create Temp C-State Packages for C1,C3, C6, C7 and initialize them to IO LVL C-state defaults
2. Create Flags (AC2V,AC3V) indicate whether ACPI C2 and ACPI C3 packages are valid/invalid
First call to _CST
YES
Update the Latency values of C3,C6,C7 temp packages from PPM NVS area
Update the IO LVL and Power Values for C7 temp Package
Is Processor and OS suports Mwait Extentions ?
YES
1. Update the ResourceTemplate of Temp packages C1,C3,C6,C7 to Mwait ext format
2. Update the Mwait Hint values of C3,C6,C7 from PPM NVS
Is Processor supports Mwait Ext and OS supports Mwait for C1 only
YES
Update C1 temp package Resource Template
NO
Initialize both ACPI C2 and C3 valid flags AC2V,AC3V to False
Copy C1 temp package to C3ST[1]
Is C3 supported
YES
Copy C3 temp package to C3ST[2]
Set AC2V flag to valid to indicate C3ST[2] is valid
If (BAT or Desktop) and Deep C-state enabled)
No
Is C6 supported?
YES
Copy C6 temp package to C3ST[3]
Set AC3V flag to valid to indicate C3ST[3] is valid
System in AC mode?
YES
Set AC2V flag to Invalid to indicate C3ST[2] is invalid as we need to report only C1,C6
YES
Copy C7 temp package to C3ST[3]
Set AC3V flag to valid to indicate C3ST[3] is valid
NO


LakeTiny GEAR2 ?
NO
LakeTiny GEAR3 ?
YES
Is C3 supported?
NO
Copy C1 temp package to C3ST[2] and C3ST[3]
YES
Copy C3 temp package to C3ST[2] and C3ST[3]
Is both AC2V and AC3V are valid
YES
Return C3ST
Is AC2V Valid?
YES
1. Copy C3ST[1] to C2St[1]
2. Copy C3ST[2] to C2St[2]
3.Return C2ST
Is AC3V valid?
YES
1. Copy C3ST[1] to C2St[1]
2. Copy C3ST[3] to C2St[2]
3. Update the C2ST[2] package C-state number to 2.
4. Return C2ST
Copy C3ST[1] to C1ST[1].
Return C1ST
End

3.2.3 Modules

3.2.3.1 PowerMgmtInit Dxe Driver

3.2.3.1.1 Introduction

The Power Management Init Dxe driver is the main entry point of the PPM Reference Code. The driver performs tasks such as setting the PpmNvsArea, acquiring configuration policies, initializing PPM features based on processor capabilities and configuration policies, registering SMM handler for restoring MSR values during S3, generating ACPI tables and Initializing the IA package and DDR RAPL settings.

The driver is configured via a CPU Dxe Platform Policy protocol -> PowerMgmtConfig. It provides platform configuration preferences for the PowermgmtInit driver to use in enabling and performing Power Management functions.

The policy protocol must be produced by a platform component prior to PPM code execution.

The Power Management Init DXE driver assumes BIOS_RESET_CPL bit is set. (SA RC set BIOS_RESET_CPL in SA PEIM)

3.2.3.1.2 Prerequisites

- Basic processor initialization must be complete
  - SMP/CMP processor initialization must be complete
  - PPM and thermal MSR must not have been locked by prior code execution
  - `EFI_DEVICE_PATH_PROTOCOL` – Documented in *EFI Specification 2.3*
  - `EFI_LOADED_IMAGE_PROTOCOL` – Documented in *EFI Specification 2.3*
  - `EFI_ACPI_SUPPORT_PROTOCOL` – Documented in the *Intel® Platform Innovation Framework for EFI ACPI Specification*
  - EFI_ACPI_TABLE_PROTOCOL – Documented in PI Specification 1.2

3.2.3.1.3 Result

The system Power management settings are determined and set up and the PPM related ACPI tables are ready for OS to consume.

3.2.3.2 PowerMgmtS3 Driver

3.2.3.2.1 Introduction

The Platform Power ManagementS3 driver is an SMM driver whose main purpose is to save MSR values during boot and restore them during S3. We set the Restore Flag to TRUE or FALSE so that we will not try to restore the MSR entries that flagged the Restore flag as FALSE.

The driver is a Framework DXE driver designed to function primarily in SMM mode. The only normal DXE functionality is to register the driver for SMM use.

The PowerMgmtInitDone protocol must be produced by the PowerMgmtInit Driver prior to PPM code execution.

3.2.3.2.2 Prerequisites

- Basic processor initialization must be complete
- SMP/CMP processor initialization must be complete
- SMM initialization must be complete
- SMM MP services must be available
- **EFI_SMM_BASE_PROTOCOL** – Documented in *System Management Mode Core Interface Specification* (SmmCis.pdf)
- **EFI_SMM_SYSTEM_TABLE** – Documented in *System Management Mode Core Interface Specification* (SmmCis.pdf)
- **EFI_SMM_SW_DISPATCH_PROTOCOL** – Documented in *Framework Lynx Point Reference Code Design Specification*

3.2.3.2.3 Result

The system PPM settings are determined and set up, the PPM related SMM handlers are functional, and the PPM related ACPI tables are ready for OS to consume.

3.2.3.3 ACPI Tables Data File

3.2.3.3.1 Introduction

The PPM specific ACPI Tables are provided as ACPI SSDT tables that are converted into a Framework compliant data file that is located and loaded by the PowerMgmtInit driver. The tables are published using Framework services to generate appropriate ACPI objects for consumption by OS. The Power management Code publishes Processor power Management related ACPI objects that

1. Report C-States
2. Report P-States
3. Report T-States
4. Dynamically load C-State and P-State support depending on configuration and OS driver capabilities.

3.2.3.3.2 Prerequisites

- CFGD: It is mirror value of PPM flags
- _PDC: Collect OS driver capabilities and dynamically loads C-State/P-State tables
- PpmGlobalNvs: Ppm Global ACPI NVS buffer used as a communications buffer between Dxe code and ASL code

3.2.3.3.3 Result

Following ACPI methods are by the ACPI Tables ASL

- _OSY: The value of OS type
- _PPC, _PCT, _PSS: Publish P-State capabilities
- _CST: Publish C-State capabilities
- _TPC, _PTC, _TSS, _TSD: Publish T-State capabilities

Table 1. Intel _PDC Definitions

Bit [0]	***OS capable of access to IA32_PERF_CTL via FFH in _PCT***
Bit [1]	***OS capable of supporting "I/O; HALT" sequence for C1 handler***
Bit [2]	***OS capable of access to IA32_THERM_CTL via FFH in _PTC***
Bit [3]	***OS capable of supporting _CST with only C1 in MP*** ***OS capable of supporting _PSS, _TSS for P-States and T-State in MP***
Bit [4]	***OS capable of supporting _CST with C2/C3 in MP***
Bit [5]	***OS capable of P State Software Coordination via _PSD***
Bit [6]	***OS capable of C State Software Coordination via _CSD***
Bit [7]	***OS capable of T State Software Coordination via _TSD***
Bit [8]	***OS capable of using Monitor/MWAIT for C1***
Bit [9]	***OS capable of using Monitor/MWAIT for C2/C3***

3.2.3.4 PPM GLOBAL NVS AREA PROTOCOL

Summary

This protocol provides definition of global NVS area protocol.

GUID

```
#define EFI_PPM_GLOBAL_NVS_AREA_PROTOCOL_GUID \
  {0x6c50cdcb, 0x7f46, 0x4dcc, 0x8d, 0xdd, 0xd9, 0xf0, 0xa3, 0xc6, 0x11,
0x28 \
  }
```

Protocol Interface Structure

```
typedef struct _ PPM_GLOBAL_NVS_AREA_PROTOCOL {
EFI_GLOBAL_NVS_AREA     *Area;
} EFI_GLOBAL_NVS_AREA_PROTOCOL;
```

Parameters

Area

Pointer to a PPM global NVS area.

Description

This protocol provides PPM specific global NVS area definitions. The memory pointed to must be of type ACPI NVS (see ACPI and UEFI specification for details on allocating NVS memory).

Related Definitions

None

PPM_GLOBAL_NVS_AREA Fields

Summary

This section describes the PPM specific Global NVS Area fields that are referenced by the Power Management Module. The PPM Global NVS Area is the communication buffer between PPM code and ASL code. Above PPM Global NVS Area fields are part of Power management code

3.3 Thermal Reporting

Thermal Reporting Module, a.k.a. DTS (Digital Thermal Sensor), is a Framework compliant DXE driver and support libraries. This driver is configured via the DTS item in CPU Platform Policy protocol. The item must be produced by a platform component prior to this module's initialization. Through policy protocol DTS driver can be configured for temperature reporting via threshold interrupts or just enabling the critical temperature interrupt for graceful shutdown of the system in the event of Out of Spec temperature condition. The whole implementation is based on *Haswell Processor family BIOS Writer's Guide*.

3.3.1 Thermal Reporting (DTS) Module Execution Flow

1. Locate CPU Policy Protocol and Initialize SMM Driver
2. Copy DTS driver to SMM RAM
3. Generate SMI for SMM dispatching initialization

4. Initialize the global variables
5. Initialize the support libraries
6. Register the callback function based on the DTS item in CPU platform policy
7. Enable DTS by setting appropriate MSR bits
8. Initialize DTS Code Definitions

3.3.2 Interfaces and Functions

This section describes code definitions for the DTS functions in this module.

3.3.2.1 Dependencies

- ***EFI_LOADED_IMAGE_PROTOCOL*** - Documented in *EFI Specification 1.10* (EFI_1-10.pdf)
- ***EFI_ACPI_SUPPORT_PROTOCOL*** - Documented in the Intel® Platform Innovation Framework for EFI ACPI Specification.
- ***EFI_SMM_BASE_PROTOCOL*** - Documented in *System Management Mode Core Interface Specification* (SmmCis.pdf)
- ***EFI_GLOBAL_NVS_AREA_PROTOCOL*** - Documented in the Framework Platform Power Management Client Reference Code Architecture Specification
- ***EFI_SMM_IO_TRAP_DISPATCH_PROTOCOL*** - Documented in Cougar Point Framework BIOS Reference Code Specification
- ***EFI_SMM_SX_DISPATCH_PROTOCOL*** - Documented in Cougar Point Framework BIOS Reference Code Specification
- ***CPU_PLATFORM_POLICY_PROTOCOL*** – This Document

3.3.2.2 InstallDigitalThermalSensor ()

Summary

This procedure is the driver entry point for the DXE driver.

Prototype

```
EFI_STATUS
InstallDigitalThermalSensor (
  IN EFI_HANDLE             ImageHandle,
  IN EFI_SYSTEM_TABLE       *SystemTable
);
```

Parameters

ImageHandle

The firmware allocated handle to the Driver Image

SystemTable

Pointer to the EFI System table

Description

This procedure checks platform policy item (“EnableDts”) setting and initializes/installs corresponding SMI events and event handlers.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.3 InitializeDtsHookLib ()

Summary

Place to call OEM hooks.

Prototype

```
EFI_STATUS
InitializeDtsHookLib (
  VOID
);
```

Parameters

None

Description

Put OEM code here.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.4 DigitalThermalSensorEnable ()

Summary

Initializes the Thermal Sensor Control MSR. This function must be AP safe.

Prototype

```
EFI_STATUS
DigitalThermalSensorEnable (
  VOID              *Buffer,
);
```

Parameters

Buffer

Unused in this function.

Description

This procedure enables DTS.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.5 DigitalThermalSensorInit ()

Summary

Performs first time initialization of the Digital Thermal Sensor, based on platform DTS policy to enable threshold table interrupts or critical temperature interrupt for out of spec condition

Prototype

```
EFI_STATUS
DigitalThermalSensorInit (
  VOID
);
```

Parameters

None

Description

This procedure does all the DTS initialization.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.6 DtsSmiCallback ()

Summary

SMI handler to handle DTS CPU Local APIC SMI for thermal threshold interrupts.

Prototype

```
EFI_STATUS
DtsSmiCallback (
  EFI_HANDLE      SmmImageHandle,
  VOID            *CommunicationBuffer,
  UINTN           *SourceSize,
);
```

Parameters

SmmImageHandle

The firmware allocated handle to the Driver Image

CommunicationBuffer

Pointer to the buffer that contains the communication Message

SourceSize

Size of the memory image to be used for handler

Description

This procedure handles DTS SMI events and depending on processor capability, it will either handle per-core DTS SMI or per-package DTS SMI events. Per-package DTS SMI will be chosen and initialized by default if the processor supports it.

Status Codes Returned

EFI_SUCCESS	Init Digital Thermal Sensor successfully

3.3.2.7 DtsIoTrapCallback ()

Summary

This catches IO trap SMI generated by the ASL code to enable the DTS AP function

Prototype

```
EFI_STATUS
DtsIoTrapCallback (
  EFI_HANDLE      DispatchHandle,
  EFI_SMM_IO_TRAP_DISPATCH_CALLBACK_CONTEXT
        *CallBackContext
);
```

Parameters

DispatchHandle

Not used in this function.

CallbackContext

Not used in this function.

Description

This procedure does necessary DTS initialization for system Standby entry and exit, and this callback function is required to be called in ACPI _PTS() and _WAK() methods.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.8 DtsS3EntryCallback ()

Summary

Callback function for S3 Entry to clear DTS status flags

Prototype

```
EFI_STATUS
DtsS3EntryCallback (
  EFI_HANDLE      Handle,
  EFI_SMM_SX_DISPATCH_CONTEXT                    *Context,
);
```

Parameters

Handle

Handle of the callback.

Context

The dispatch context.

Description

This procedure does all the DTS initialization required for Standby entry.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.9 DtsOutOfSpecSmiCallback ()

Summary

SMI Handler to handle DTS CPU Local APIC SMI for Critical Temperature Interrupts

Prototype

```
EFI_STATUS
DtsOutOfSpecSmiCallback (
  EFI_HANDLE      SmmImageHandle,
  VOID                    *CommunicationBuffer,
  UINTN                   *SourceSize
);
```

Parameters

SmmImageHandle

The firmware allocated handle to the Driver Image

CommunicationBuffer

Pointer to the buffer that contains the communication Message

SourceSize

Size of the memory image to be used for handler

Description

This procedure handles the DTS Out-of-spec SMI event. This SMI event and handler will be initialized when the "EnableDts" field in platform policy is set to 2 (Out of Spec Only). From reference code 0.8.0 this "Out of Spec Only" DTS operation mode has been separated from DTS Enable mode. When the temperature has reached a critical temperature threshold, the SMI handler will set the DTS operation mode to DTS_OUT_OF_SPEC_OCCURRED (3), to indicate a critical temperature condition has been reached and it is required to do a graceful system shutdown immediately.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

3.3.2.10 DTS Item of CPU Platform Policy Protocol

Summary

This item in CPU policy protocol is for DTS to pass in user selectable DTS options to DTS driver.

Definition

EnableDts

The field describes the selected DTS policy –

- "Disable" (set to 0 - default)
- "Enable" (set to 1)
- "Critical Temp Reporting (Out of spec)" (set to 2)

Description

The data elements should be initialized by a Platform Module. The data structure is located through its GUID to retrieve CPU platform policy.

ACPI GlobalNvs -> EnableDigitalThermalSensor will be initialized by DTS module to indicate current DTS operation mode and ACPI thermal zone scope should check this field and use proper thermal reporting mechanism depending upon different DTS operation mode.

ACPI GlobalNvs -> EnableDigitalThermalSensor field will have following value set:

DTS_SMM_DISABLE (0) – This value will be set when the platform policy "EnableDts" is set to "Disable" (0). No DTS SMI events will be enabled and no DTS SMI event handler will be installed. In this case, an alternate thermal reporting mechanism should be applied in ACPI thermal zone instead of DTS.**DTS_SMM_ENABLE (1)** – This value will be set when the platform policy "EnableDts" is set to "Enable" (1). DTS threshold interrupt SMI events will be enabled

and corresponding event handler will be installed. ACPI thermal zone may use DTS as thermal reporting mechanism.

DTS_OUT_OF_SPEC_ONLY (2) – This value will be set when the platform policy "EnableDts" is set to "Critical Temp Reporting (Out of spec)" (2) and the critical temperature has not been reached. Unlike the "Enable" mode, the DTS threshold interrupt SMI events will not be enabled thus DTS temperature data will not be updated during runtime. Only the "critical temperature interrupt" SMI event will be enabled and this SMI event will be generated when the temperature has reached a factory configured critical temperature. From reference code 0.8.0 the "Critical Temp Reporting (Out of spec)" mode has been separated from "Enable" mode and ACPI thermal zone should implement alternate thermal reporting mechanism when ACPI GlobalNvs -> EnableDigitalThermalSensor is not 1 (DTS "Enable" mode).

DTS_OUT_OF_SPEC_OCCURRED (3) – This is an indicator to ACPI thermal zone to request a graceful shutdown because processor has reached critical temperature condition. When DTS operation mode is "Critical Temp Reporting (Out of spec)" (set platform policy "EnableDts" to 2) and processor temperature has reached Out-of-spec condition, this value will be set to ACPI GlobalNVS "EnableDigitalThermalSensor". ACPI thermal zone should report critical trip point temperature to OS and have OS do graceful shutdown immediately when this value set to ACPI GlobalNvs -> EnableDigitalThermalSensor.

3.4 TXT

This specification is useful for any software developer using an Intel® Trusted Execution Technology (Intel® TXT) component and a Framework compliant system BIOS that utilizes the reference code for Haswell platform.

This document describes the component and its software interfaces in sufficient detail for system BIOS developers to use and modify the Intel TXT reference code.

3.4.1 Overview

3.4.1.1 Architectural Overview

TxtInit PEIM

The main task of this module is to determine the status of the memory. If the memory is locked, TXT PEIM prepares the environment to execute BIOS ACM and then calls its SCLEAN function. The SCLEAN function unlocks the memory so that upon reset a normal boot will occur. Changes in section 6.2.5 of *RS – Intel Trusted Execution Technology BIOS Specification*, SCLEAN on Haswell platform which indicate SCLEAN will only unlock memory and clear secrets flag, it's BIOS's responsibility to scrub memory and make sure contents of memory are over-written.

TxtInit DXE Driver

The main task of this driver is to run SCHECK of BIOS ACM and to program Intel TXT heap memory and registers in Intel TXT public space as required by *RS – Intel Trusted Execution Technology BIOS Specification*. The SCHECK function verifies the correctness of the memory configuration and locks it. The information in the heap memory and public space registers is used by the system software to establish the MLE.

AP Initialization Module

`AP Initialization Module` conceptually is part of `TXT PEIM` since it works together with it for preparation of the SCLEAN launch environment but architecturally is separated because it must reside in flash part on 4KB boundary. This requirement is imposed by the SIPI vector creation rule.

The main tasks of the `AP Initialization Module` consist of the following — All APs' CR0.CD and CR.NW must be ensured clear with cache enabled; CPU micro code patch must be loaded; MCA registers must be cleaned. Full details can be found in *RS – Intel Trusted Execution Technology BIOS Specification*.

BIOS ACM

The BIOS AC Module is a chipset specific signed binary provided by Intel and is called to perform functions required to enable the Intel TXT environment. The AC module is loaded into and executed from the CPU cache. This area of the CPU cache is known as the Authenticated Code RAM (AC RAM).

The `BIOS AC Module` is launched by the BIOS to perform the following specific functions.

- Unlock Memory (SCLEAN) — Unlocks the memory by writing through TXT private space registers.
- Reset TPM Establishment Flag — Resets the TPM establishment flag to factory default.
- Check and Register (SCHECK) — Checks the current version of the BIOS AC module that is registered in the TPM NV RAM.

3.4.1.2 Dispatch Flow

According to aforesaid architectural description, Intel TXT sub-components and support modules must be dispatched in the following order.

```
Platform PEIM | TCG/TPM PEIM ->

TXT PEIM ->

Memory Initialization PEIM | CPU PEIM ->
```

Platform DXE -> TXT Driver

Platform PEIM and TCG/TPM PEIM can be mutually executed in any order but both must be dispatched before TXT PEIM in order to properly create TXT Info Hob and initialize TPM.

TXT PEIM must be dispatched before Memory Initialization PEIM in order to unlock memory interface if necessary.

Memory Initialization PEIM and CPU PEIM can be mutually executed in any order, but CPU PEIM must be dispatched at least after Platform PEIM since it consumes TXT Info Hob information.

Platform DXE driver initiates TxtPolicyProtocol which is executed by TxT DXE driver.

TXT Driver is executed in DXE phase and is dispatched last.

3.4.2 Code Definitions

3.4.2.1 TxtInit PEIM

Description

Bit 0 of TPM Command Register located at address 0xFED40000 has special meaning for Intel TXT enabled chipset. This bit is called "Establishment" bit and initially has value logic "1". When MVMM is started on the platform the first time, Establishment bit is asserted to the value logic "0".

During a platform boot, read access to Establishment bit is snooped by chipset. If bit is deasserted (logic "1") the fact of the read itself will unconditionally unlock the memory interface. This is why memory initialization code is required to perform a read of the byte containing this bit.

If Establishment bit is asserted (logic "0"), chipset performs further checking. If it "thinks" that there are no secrets present in memory – chipset unlocks the memory interface and boot proceeds normally. Otherwise, if secrets are present in memory,

the memory interface remains locked and the only way to unlock it for BIOS is to execute the SCLEAN function of `BIOS ACM`.

Software can determine the locked memory status by checking the "Wake Error Status" bit – bit 6 of Status Register located at address 0xFED40008. If Wake Error Status bit equals 1 – memory is locked.

The above description stipulates the main task of the `TxtInit PEIM` – it must detect whether memory is locked. To do so, it is necessary to call the SCLEAN function of `BIOS ACM`.

If SCLEAN has to be run, `TxtInit PEIM` prepares the environment as specified in section 6.2.5 of *RS – Intel Trusted Execution Technology BIOS Specification* – it starts-up and initializes APs, puts APs in WFS state, prepares BSP to launch ACM, and finally executes the `GETSEC[ENTERACCS]` instruction with input parameters directing ACM to execute the SCLEAN function.

Dependencies

PEI_STALL_PPI_GUID — this PPI is necessary because `TXT PEIM` uses **`PEI_STALL_PPI`** services

PEI_TPM_INITIALIZED_PPI_GUID — this PPI is installed by `TCG PEIM` and signals that TPM PEIM has been loaded.

PEI_CPU_PLATFORM_POLICY_PPI_GUID — Policy PPI which is installed by Platform code.

Interface

`PEI_TXT_MEMORY_UNLOCKED_PPI`

Summary

This PPI is installed in the beginning of TXT PEIM Entry Code in order to signal to Memory Initialization PEIM that the memory interface is unlocked.

PPI is installed always, regardless of TXT enabling status. This guarantees that Memory Initialization PEIM will run.

GUID

```
#define PEI_TXT_MEMORY_UNLOCKED_PPI_GUID \
{0x38cdd10b,0x767d,0x4f6e,0xa7,0x44,0x67,0xee,0x1d,0xfe,0x2f,0xa5}
```

PPI Interface Structure

```
EFI_GUID  gPeiTxtMemoryUnlockedPpiGuid =
  PEI_TXT_MEMORY_UNLOCKED_PPI_GUID;
static EFI_PEI_PPI_DESCRIPTOR mPpiList = {
  (EFI_PEI_PPI_DESCRIPTOR_PPI | EFI_PEI_PPI_DESCRIPTOR_TERMINATE_LIST),
  &gPeiTxtMemoryUnlockedPpiGuid,
  NULL
};
```

Parameters

PPI Interface structure is empty and doesn't have any control methods.

Description

PPI is used only to signal subsequent modules that TXT PEIM has been run.

Related Definitions

None

3.4.2.2 TxtInit DXE Driver

Description

The TxtInit DXE driver is a major sub-component of Intel TXT component. It collects all information passed to it through TXT Info Hob and performs the following:

- It determines whether Intel TXT is enabled by reading CPU IA32_FEATURE_CONTROL_MSR.

- If Intel TXT is enabled, TxtInit DXE Driver first programs TXT Heap Memory and Intel TXT registers in Intel TXT public space, as required by *RS – Intel Trusted Execution Technology BIOS Specification*. Along with this programming TXT Driver adds respective register values to Boot Script. This will ensure the restoring of register values upon resume from S3.
- It then creates a "ReadyToBoot" event and in the context of the callback function it executes the SCHECK function of BIOS ACM. The main purpose of SCHECK is to validate the correctness of the platform configuration, lock this configuration, and check and update BIOS ACM registration in TPM NV RAM. In order to launch BIOS ACM TXT Driver code programs BSP and APs in the way analogous to the TXT PEIM.
- If Intel TXT is disabled but Establishment bit is asserted (logic "0"), in order to avoid possible "brick" situation when BIOS and / or BIOS ACM is updated, TXT Driver runs BIOS ACM RESET_ESTABLISHMENT_BIT function. The procedure is skipped if TPM doesn't get initiated first.
- TxT DXE driver reads policy settings from CpuPlatformPolicyProtocol. For now, ResetAux is the only function. Once Platform DXE driver sets ResetAux of the policy protocol, TxtInit DXE driver will lunch ResetAux function to clear Auxiliary content.

Dependencies

EFI_BOOT_SCRIPT_SAVE_PROTOCOL_GUID — TxT DXE driver saves registers through this protocol for S3 resume.

EFI_MP_SERVICES_PROTOCOL_GUID — The protocol provides the services for initiate CPUs.

EFI_SMM_BASE_PROTOCOL_GUID — TxT DXE driver requires to be loaded after SMM Base driver is loaded.

EFI_CPU_IO_PROTOCOL_GUID — Access MMIO/IO registers.

DXE_CPU_PLATFORM_POLICY_PROTOCOL_GUID — Policy protocol which is installed by Platform code.

Interface

None

Related Definitions

None

3.4.2.3 AP Initialization Module

Description

- AP Initialization Module requires special placement in flash part – specifically the code section of TxtPeiAp.FFS file must be located on 4KB boundary. This is because AP initialization code is invoked as a result of the SIPI message generated by TxtInit PEIM.

  The SIPI message contains an 8 bit vector field describing message destination. Physical address where destination startup code must be located is calculated by CPU as Vector << 12 and all low order bits are assumed to be 0.

  The formula above stipulates that the highest address where the destination code can be placed equals to `0xFF << 12 = 0xFF000` – address that is below 1 MB.

  From another standpoint only two topmost blocks of flash part are reflected to addresses below 1 MB. This limits the range where the `TXTPEIAP.FFS` file must be placed – between addresses `0xE0000 and 0xFF000`.

- `AP Initialization Module` must be compiled as a standalone file. This requirement is imposed by the aforementioned 4-KB alignment. If `AP Initialization Module` were compiled with PE or TE header then when `TXTPEIAP.FFS` is placed into flash part, FV creation tools would force header to be located on 4-KB boundary and since header size is not equal to 4 KB, code section of `TXTPEIAP.FFS` file would be shifted off 4-KB boundary.

- `AP Initialization Module` needs a special fix-up procedure. Since it is compiled as a standalone file without header, build tools don't have the fix-up table needed for fix-ups.

  As a result, all offsets inside of `AP Initialization Module` remain relative to the beginning of the file. At the same time, code in AP Initialization Module needs

access to parameters passed through registers located in Intel TXT public space in high memory. For such access AP Initialization code needs to run in big real or protected mode. Switching to any of these modes is impossible without fixed-up variables embedded into module's code. This dilemma is solved by using the separate fix-up tool `STAFIXUP.EXE` run as part of build process.

Prerequisites

TxtPeiAp.FFS file must be located on 4KB boundary

`TXTPEIAP.FFS` file must be placed – between addresses `0xE0000 and 0xFF000`

Related Definitions

AP Initialization Module FFS File GUID

```
#define PEI_AP_STARTUP_FILE_GUID \
 { \
   0xD1E59F50, 0xE8C3, 0x4545, 0xBF, 0x61, 0x11, 0xF0, 0x02, 0x23, 0x3C, 0x97 \
 }
```

3.4.2.4 BIOS ACM

Description

BIOS ACM interface is described in RS – Intel Trusted Execution Technology BIOS Specification

Prerequisites

BIOS ACM must be placed in flash part on 4KB boundary

Related Definitions

BIOS ACM FFS file GUID

```
#define PEI_BIOS_ACM_FILE_GUID \
 { \
   0x2D27C618, 0x7DCD, 0x41F5, 0xBB, 0x10, 0x21, 0x16, 0x6B, 0xE7, 0xE1, 0x43 \
 }
```

3.4.2.5 TpmInitialized PPI

Description

PEI_TPM_INITIALIZED_PPI_GUID is installed by TCG/TPM PEIM if TPM module is initiated successful..

Definitions

```
#define PEI_TPM_INITIALIZED_PPI_GUID \
 { \
   0xe9db0d58, 0xd48d, 0x47f6, 0x9c, 0x6e, 0x6f, 0x40, 0xe8, 0x6c, 0x7b, 0x41 \
 }
```

3.4.2.6 TxtOneTouch GUID

Description

TxT One Touch is optional. Please refer to TxT One Touch BIOS spec for detail information and refer to TCG Physical Presence specification.

The GUID includes the definitions which are defined by TxT One Touch BIOS spec.

Definitions

```
#define TXT_ONE_TOUCH_GUID \
  { \
```

```
    0x3D989471, 0xCFAC, 0x46B7, 0x9B, 0x1C, 0x8, 0x43, 0x1, 0x9, 0x40,
0x2D \
  }


#define  ENABLE_VT                              128
#define  DISABLE_VT_TXT                         129
#define  ENABLE_VTD                             130
#define  DISABLE_VTD_TXT                        131
#define  ENABLE_ACTTPM_VT_VTD_TXT_DISABLE_STM   132
#define  ENABLE_ACTTPM_VT_VTD_TXT_STM           133
#define  DISABLE_STM                            134
#define  DISABLE_TXT_STM                        135
#define  DISABLE_SENTER_VMX                     136
#define  ENABLE_VMX_SMX_ONLY                    137
#define  ENABLE_VMX_OUTSIDE_SMX                 138
#define  ENABLE_VMX                             139
#define  ENABLE_SENTER_ONLY                     140
#define  ENABLE_SENTER_VMX_IN_SMX               141
#define  ENABLE_SENTER_VMX_OUTSIDE_SMX          142
#define  ENABLE_SENTER_VMX                      143
#define  SET_NO_TXT_MAINTENANCE_FALSE           159
#define  SET_NO_TXT_MAINTENANCE_TRUE            160
```

Prerequisites

BIOS must support Physical Presence function.

3.4.2.7 TxtInfo HOB GUID

Description

TxtInfo HOB is used for passing platform policy settings and CPU/chipset information within TxT modules.

Definitions

```
#define TXT_INFO_HOB_GUID \
{0x2986883F,0x88E0,0x48d0,0x4B,0x82,0x20,0xC2,0x69,0x48,0xDD,0xAC}

typedef struct {
  BOOLEAN                   ChipsetIsTxtCapable;
  UINT8                     TxtMode;
  UINT64                    PmBase;
  UINT64                    SinitMemorySize;
  UINT64                    TxtHeapMemorySize;
  EFI_PHYSICAL_ADDRESS      TxtDprMemoryBase;
  UINT64                    TxtDprMemorySize;
  EFI_PHYSICAL_ADDRESS      BiosAcmBase;
  UINT64                    BiosAcmSize;
  EFI_PHYSICAL_ADDRESS      McuUpdateDataAddr;
  EFI_PHYSICAL_ADDRESS      SinitAcmBase;
  UINT64                    SinitAcmSize;
  UINT64                    TgaSize;
  EFI_PHYSICAL_ADDRESS      TxtLcpPdBase;
  UINT64                    TxtLcpPdSize;
  UINT64                    Flags;
} TXT_INFO_DATA;

#define FLAGS0                  0x1
#define TXT_CPU_RESET_REQUIRED  0x2
#define TPM_INIT_FAILED         0x4
```

Parameters

ChipsetIsTxtCapable	Boolean value is set to logic "1" if chipset is Intel® TXT capable.
TxtMode	Boolean value is set to logic "1" if Intel TXT mode is enabled in BIOS Setup.
PmBase	Address of `PM1a_CNT_BLK` register block. Is used by TXT PEIM to clean Sleep Type field of `PM1a_CNT_BLK.S4` register before running of SCLEAN.
SinitMemorySize	Size of memory reserved for placement of SINIT module. This memory is used by MLE.
TxtHeapMemorySize	Size of memory reserved for TXT Heap. This memory is used by MLE.
TxtDprMemoryBase	Base address of DPR protected memory reserved for Intel TXT component.
BiosAcmBase	Base address of BIOS ACM in flash part. It can be passed through platform code for customization; Intel TXT reference code would skip searching the BIOS ACM in PEI firmware volume if the field is not zero.
BiosAcmSize	Size of BIOS ACM.
McuUpdateDataAddr	Base address of CPU micro code patch loaded into BSP. It can be passed through platform code for customization; Intel TXT reference code would skip searching the micro code path in PEI firmware volume if the field is not zero.
SinitAcmBase	Base address of SINIT module if installed in flash part. Zero otherwise.
SinitAcmSize	Size of SINIT module if installed in flash part. Zero otherwise
TgaSize	Size of *Trusted Graphics Aperture* if supported by chipset. For Cantiga must be 0.
TxtLcpPdBase	Base address of *Platform Default Launch Control Policy* data if installed in flash part. Zero otherwise.
TxtLcpPdSize	Size of *Platform Default Launch Control Policy* data if installed in flash part. Zero otherwise.
Flags	Up to 64 bit flags passed from BIOS to OS or MRC BIT0 - FLAGS0 for compatible definition BIT1 - TXT_CPU_RESET_REQUIRED for MRC to issue reset if required BIT2 - TPM_INIT_FAILED for indicate TPM initiate status. If the bit set, ResetEstablishmentBit is skipped in Dxe driver.

3.4.3 STAFIXUP Tool

Building of the TXT component requires the fixing up of the AP Initialization Module file. This task is performed by the custom STAFIXUP utility.

Description

STAFIXUP gets, as one of the command line parameters, the GUID of the FFS file. It looks for this GUID in the binary image of BIOS ROM file.

It takes, as a second parameter, the name of the text file which has the fix-up information. This is analogous to the fix-up table of the PE header. Specifically, each line of text file contains the following information.

Offset from beginning of code section to be fixed;

Width of data to be fixed.

Supported widths are 2 and 4 bytes.

Finally it takes as a parameter range of BIOS ROM file to be searched.

STAFIXUP then searches a specified range of offsets and after the FFS file with a specified GUID is found, it performs its fix-up using the following methodology.

It assumes that the end of the BIOS ROM file corresponds to physical address 4 Gb for each 4 bytes-wide fix-up and 1 Mb for each 2 byte-wide fix-up.

It then computes the physical address of the beginning of the code section of the found file by skipping its header.

It reads from the text file fix-up offsets and widths and performs fix-ups using the following formula:

If (width = 2 bytes)

New_Content_Of_WORD_At_Given_Offset =

Old_Content_Of_WORD_At_Given_Offset +

1MB – Offset_from_File_End

Else if (width = 4 bytes)

New_Content_Of_DWORD_At_Given_Offset =

Old_Content_Of_DWORD_At_Given_Offset +

4GB - Offset_from_File_End

Else

Error

It checksums the file if it is requested by the file attributes

Fix-Up File Format Sample

```
#
# File format:
#-----------------
# Offset, Width
#
0x12,    4
```

Command Line Interface

When called without parameters STAFIUP displays the following command line help.

Standalone file fix-up tool. Rev. 1.0, Copyright (c) Intel Corp. 2007

STAFixup <GUID> <EfiFileName> <FuFileName> [<StartOffset> [<EndOffset>]]

GUID - GUID of FFS file to fix-up.

EfiFileName - EFI Image file containing the above FFS file.

FuFileName - Text file containing fix-up table.

StartOffset - Offset from end-of-file to start search from.

EndOffset - Offset from end-of-file to stop search at.

First three parameters – GUID, EfiFileName and FuFileName represent the GUID of FFS file to search for; BIOS ROM file name to search and Fix-up table file name.

These parameters are mandatory and their usage is self-explanatory.

The other two parameters – StartOffset and EndOffset specify the range of the search. Both offsets are computed from the end of file. These parameters are optional. If both are omitted, the entire BIOS ROM file will be searched. If only StartOffset is specified, the search will start of offset (End_Of_File – StartOffset) and will continue until the end of the file. If both are specified, only a range of offsets
((End_Of_File – StartOffset) - (End_Of_File – EndOffset)) will be searched.

Safeguarding of Build

Since, as it was explained above, AP Initialization Module must reside in flash part between 0xE0000 and 0xFF000 addresses, an ability of STAFIXUP tool to search only a given offset range can safeguard Developer against wrong placement of this module in flash part.

For instance the following example will search only the last two blocks of BIOS ROM file.

```
STAFIXUP My_GUID My_ROM My_Table 0x20000
```

If My_GUID is not found in the last two blocks, STAFIXUP will return an error and stop the build. An example is as below.

```
STAFixup.exe D1E59F50-E8C3-4545-BF61-11F002233C97 0ABOZ035.rom
Platform\IntelMpg\Common\Txt\Pei\Ia32\Apfixup.txt 0x20000

STAfixup utility for patching TXT AP PEI module address fixup, Rev. 1.0.1
Copyright (c) Intel Corp. 2007-2009

Error: FFS GUID D1E59F50 4545E8C3 F01161BF 973C2302 for TxtPeiAp.BIN has not
been found
    in file 0ABOZ035.rom by searching from end of file minus offset 131072 bytes
```

per CPU AP initialization requirement(refer to Framework TXT reference code design spec. for more detail).

3.4.4 TxT One Touch Function

TxT One Touch function is an optional function. It defines additional operators of TPM Physical Presence Interface spec. Please refer to **Intel® Trusted Execution Technology (TXT) – One-Touch Enabling spec.** There is a sample code (TxtOneTouchDxe driver) in sample code folder. The sample code shows the sample for each operator's function. BIOS developers need follow **ntel® Trusted Execution Technology (TXT) – One-Touch Enabling** and TPM Physical Presence Interface spec to develop the function.

3.5 Boot Guard

3.5.1 Overview

Haswell ULT supports Boot Guard (formerly known as Anchor Cove). Please refer to Boot Guard BWG and *Boot Guard for Shark Bay Ultrabook Architecture Overview* for detail description. With Boot Guard support, BIOS needs to have below changes.

- FIT (Firmware Interface Table) needs to report Anchor Cove ACM (Type 2).
- Report Boot Policy Manifest (Type 0x0C) in FIT
- Report Key Manifest (Type 0x0B) in FIT
- Modified NEM initialization
- Stop PBE (Protect BIOS Environment) Timer
- TPM initialization flow change

3.5.2 FIT Table

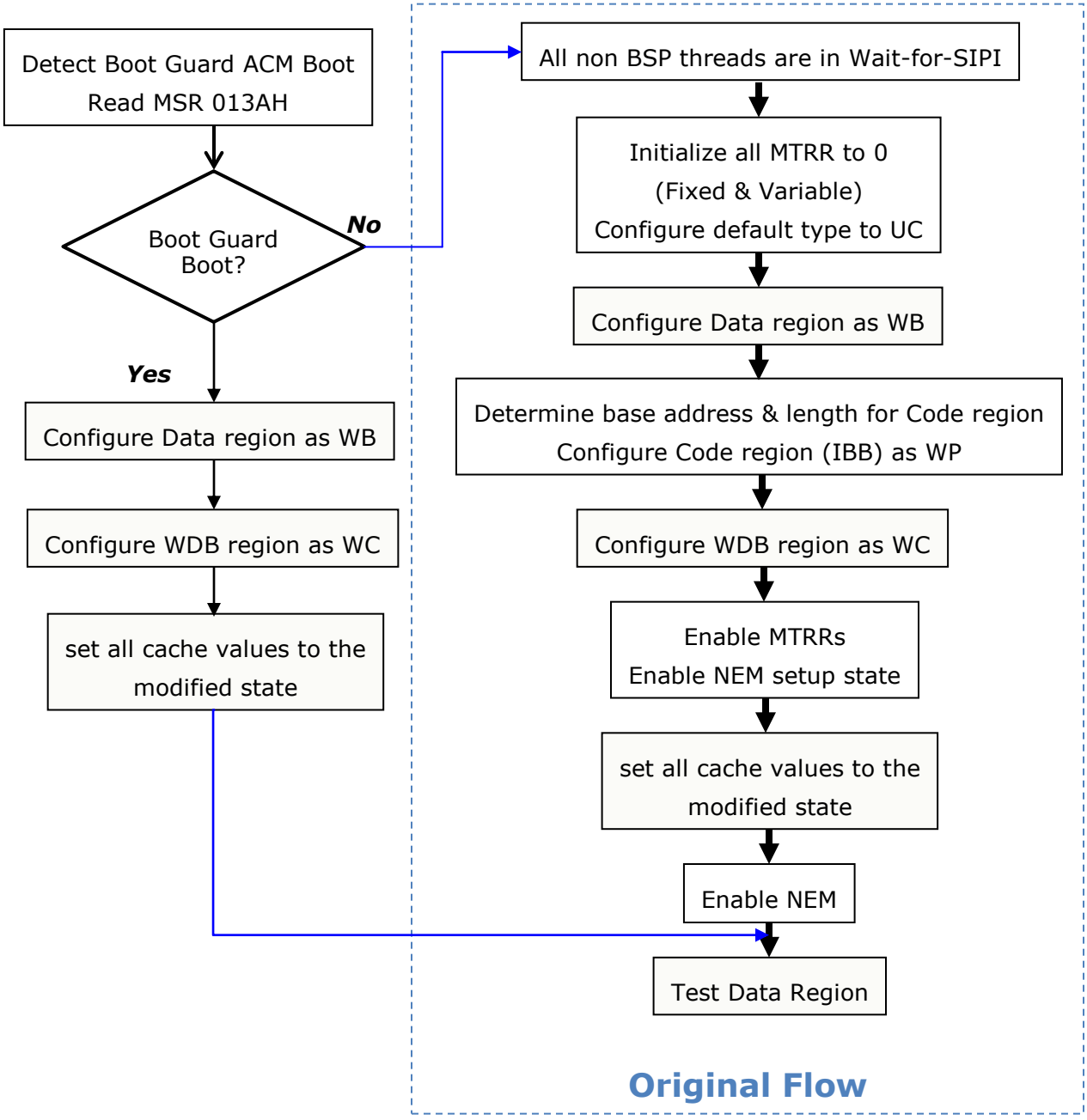
BIOS needs to implement FIT table which contains entries for uCode patch, Boot Guard ACM, Boot Policy Manifest and Key Manifest. Customers need to create a tool to generate the FIT table and fill it into BIOS image. Please refer to Boot Guard BWG and Firmware Interface Table Specification for getting detail information.

3.5.3 Boot Policy Manifest and Key Manifest

Boot Policy Manifest and Key Mainifest are required for Boot Guard. Boot Guard ACM verifies the content of Boot Policy Manifest and Key Manifest. BIOS needs to have Boot Policy Manifest and Key Manifest reported in FIT table. The structures of Boot Policy Manifest and Key Manifest are defined in Boot Guard BWG. Customers need to create a tool to generate Boot Policy Manifest and Key Manifest and fill the data into BIOS image.

3.5.4 NEM Initialization Change

Boot Guard ACM programs NEM for code region. BIOS should not program NEM again. BIOS only needs to set MTRR for covering DataStack region. The flow of NEM initialization needs to have below change. Please refer to SampleCode\SecCore.



BIOS reads bit 0 of BOOT_GUARD_SACM_INFO (MSR 0x13A) for ensure NEM enabled by ACM. If the bit is set, BIOS needs to skip NEM initialization and set MTRR for data region. Otherwise, BIOS proceeds NEM initialization flow.

3.5.5 Stop PBE Timer

When Boot Guard ACM signals it is executing, PBE Timer is started by Boot Guard ACM. The timer continues until the BSP executing the BIOS indicates to PBE Timer. BIOS needs to stop the timer before send SIPI to wake up APs. If BIOS wakes up APs without stop the timer, system gets reset. If the timer expires, system gets reset also. Haswell CPU reference code contains the code to stop PBE timer in normal boot path and S3 resume path.

3.5.6 TPM Initialization

When measured boot is set, Boot Guard ACM initialize TPM. BIOS should not send Startup command again. BOOT_GUARD_CONFIG is defined in CpuPlatformPolicy PPI. CpuInitPei checks Boot Guard status and update BOOT_GUARD_CONFIG. BIOS needs to check the policy and update TPM codes. BOOT_GUARD_CONFIG contains below information.

Item Name	Description
MeasuredBoot	TRUE – ACM performs Measured Boot FALSE – No Measured boot performed by ACM
ByPassTpmInit	TRUE – TPM initialization is done by ACM. FALSE – No TPM initialization happen in ACM
TpmType	Report what TPM is available on system. 00 – No TPM 01 – dTPM 1.2 02 – dTPM 2.0 03 - PTT
BootGuardSupport	TRUE – Boot Guard is supported FALSE –Boot Guard is NOT supported

DisconnectAllTpms	TRUE – BIOS will not to do any futher TPM initization and extends. FALSE – BIOS will continue with TPM initization based on MeasuredBoot.
ByPassTpmEventLog	TRUE – Bypass TPM Event Log if Sx Resume Type is identified. FALSE – Create TPM event log if not Sx Resume Type.

Ensure BIOS Platform code reads above status to do below items,

1. BIOS has to read DisconnectAllTpms bit to identify if all TPM's are disabled. If it is TRUE, it means CPU failed to load Boot Guard ACM and PTT or dTPM is disconnected until the next platform reset.Thus, BIOS will not do any further TPM initialization & extends.If it is FALSE, BIOS will continue with TPM initialization based on M value in Boot Guard profile.

2. With MeasuredBoot and ByPassTpmInit is TRUE, Boot Guard ACM does TPM initialization includes Startup command and extend SCRTM. BIOS will skip these two but do subsequent Extends, Platform Hierarchy and publish ACPI table. .

3. With MeasuredBoot is FALSE, Boot Guard AMC left TPM untouched, BIOS will completely initialize TPM based on BIOS policies/Setup options.

4. BIOS may have different TPM drivers for dTPM 1.2/ dTPM 2.0/PTT. TpmType reports what TPM is installed on system and initialized by Boot Guard ACM. BIOS TPM code may need to read TpmType for executing correct TPM code.

5. The ByPassTpmEventLog is indicated BIOS TPM code not to create DetailPCR or AuthorityPCR event log if Sx resume type is S3, Deep-S3, or iFFS Resume.

3.5.7 TPM Event Log

Boot Guard ACM performs measurements but within an environment without memory. Therefore Boot Guard ACM cannot create TCG Event Log Entries for the measurements. Components after Boot Guard ACM execute (e.g., IBB or DXE or even later components) must create these events. Boot Guard uses the currently defined TCG Event Log Structure currently defined for TPM 1.2 in PC Client Specification for Conventional BIOS even for PTT or dTPM 2.0. This is because Boot Guard for HSW-ULT implements SHA-1 PCRs for PTT and dTPM 2.0 and TCG has not completed definition of an Event Log Structure for TPM 2.0

The TCG defined Event Structure is copied below for reference only.

```
TCG_PCClientPCREventStruc
    pcrIndex        DD    ?
    eventType       DD    ?
    digest          DB    20 dup (?)
    eventDataSize   DD    ?
    event           DD    ?
TCG_PCClientPCREventStruc
```

If MeasuredBoot is TRUE, BIOS will responsible for calculate Digest field of Detail PCR Event or Authority PCR Event by steps below. For more detail description please refer to Boot Guard BWG and SampleCode\Library\BootGuardTpmEventLogLib.

1. Create Detail PCR Event - PCR[0]
    a. Create PCR Extend digest by using same data that ACM used to extend to PCR[0] to create an event.
        i. Find the following from FIT Table
            1. Find ACM
            2. Find KM
            3. Find BPM
            4. FindBpmIbb
            5. FindBpmSignature
        ii. Obtain Boot Policy Restrictions, Boot Policy Type, ACM SVN

iii. If MSR 0x13A indicates Verified Boot Success use the Hash value indicated in Boot Policy Manifest for IBB

iv. Create Sha1 digest of PCR Data

2. Create Authority PCR Event - PCR[6] if dTPM1.2 or PCR[7] if dTPM 2.0/PTT
   a. Create Authority PCR extend digest by using same data that ACM used to extend to authority PCR
      i. Find following from FIT Table
         1. ACM
         2. KM
      ii. Boot Policy Restrictions, Boot Policy Type, ACM SVN
      iii. Calculate ACMKeyHash

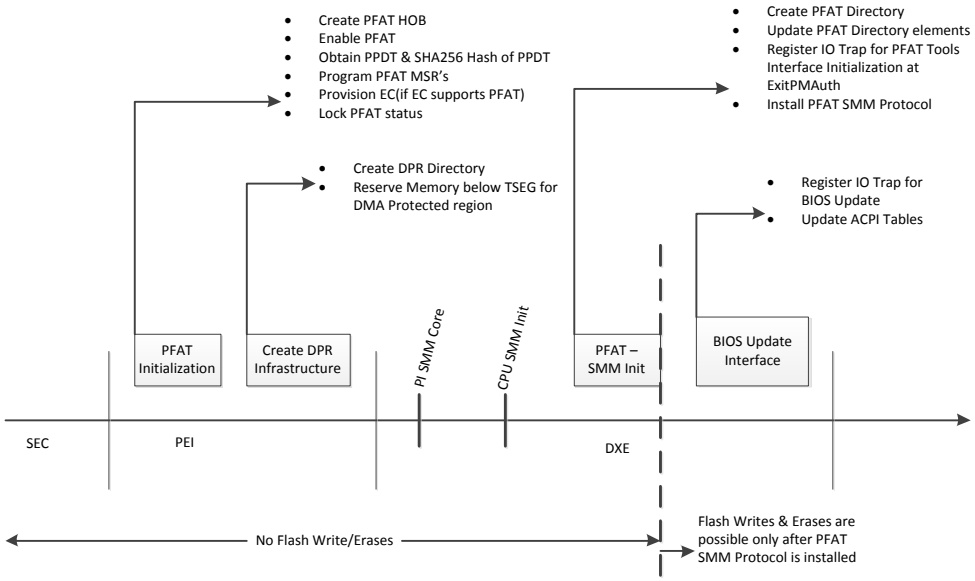
Note: The Authority PCR is not extended unless this is also a verified boot. Even if verified boot AuthorityPCR is only extended if the Boot Policy Manifest instructs it to do so.

3.6 PFAT

3.6.1 PFAT Overview

PFAT provides flash protection and BIOS update authorization using a combination of CPU and PCH features

3.6.2 PFAT Initialization Boot Flow[1]



3.6.2.1 PEI Initialization

PFAT Feature is initialized and locked based on platform policy for enabling PFAT. PFAT HOB is created for accessing relevant PFAT data in DXE, SMM phases. PFAT MSR's are programmed and locked after enabling PFAT. Here after flash writes/erases can happen only in SMM via PFAT module.

3.6.2.1.1 PPDT[2]

PFAT Platform Data Table has all the information of platform like Platform ID, Keys used for signing flash updates. Most importantly SFAM which is signed flash address map is part of PPDT. This is used to communicate to PFAT module the signed and unsigned flash regions in flash.

---

[1] RC is provided for all the components in this diagram

[2] PPDT can be static for a given platform configuration

Platform attributes like EC Present and EC support for PFAT are part of PPDT. EC Command/Status and Data ports used to communicate to EC, commands used to communicate to EC are also part of PPDT table.

SHA256 hash of PPDT is programmed into PFAT MSR's before locking PFAT status to protect the integrity of PPDT.

3.6.2.1.2 EC Support

BIOS generates ephemeral password and provisions both EC and CPU early post for PFAT support on EC. BIOS writes password to PLAT_FRMW_PROT_PASSWD for provisioning CPU. BIOS sends password to EC via KSC (Keyboard & System Management Controller) ports. EC Flash update with PFAT enabled will not use IO ports, instead will use same SMI handler in BIOS that is used to update BIOS. PUP header attributes will be read by PFAT module to identify if update package is BIOS/EC FW.

3.6.2.2 DPR Infrastructure

DMA Protected Region (DPR) is below TSEG in memory map and is shared by TXT & PFAT on platform. DPR directory[3] (part of SA RC) is created in BIOS during PEI phase and is allocated in memory map during Memory Initialization. DPR directory is in SA_DATA_HOB and contains information on layout of DPR region.

3.6.2.3 SMM Initialization

PFAT Module can be invoked in SMM only. SMM Base has to be initialized before PFAT SMM protocol is installed and any writes/erases to flash happen. During PFAT initialization in SMM

6. PFAT Directory is created

[3] Refer to Appendix in PFAT BWG for more information on DPR Infrastructure its usage by PFAT.

7. Update PFAT Directory elements with their addresses – PFAT Module & PPDT are allocated in TSEG and PUP, PUPC, & Log buffer are allocated in DPR.

8. Register IO Trap for initialization of PFAT Tools Interface. This IO trap will be triggered at ExitPMAuth[4] event allowing BIOS to proxy into SMM for initialization PFAT interface required for runtime updates.

9. Install PFAT SMM protocol.

3.6.2.4 ExitPmAuth Event

PFAT Tools interface is initialized at ExitPmAuth Event. IO Trap required for BIOS/EC FW update is registered. All PFAT ACPI tables are published with required PFAT data. Here after runtime flash updates can be done via PFAT.

3.6.3 Platform BIOS Requirements

- BIOS region SMM protection[5] must be enabled early post. This can be done based on PchPlatformPolicy. Refer to PCH Integration guide for more information.
- All UEFI variable writes/erases should happen in SMM only and after installation of SMM_PFAT_PROTOCOL.
- Enabling PFAT on platform requires all flash writes & erases to be done via PFAT Module.
- PFAT Module should be compiled into BIOS image with PFAT_MODULE_GUID

---

[4] All PFAT ACPI tables must be initialized by ExitPmAuth Event

[5] Enabling SMM Protection is a must for PFAT enabled platforms, disabling this feature will void security provided by PFAT and allows any Non-SMM driver to access flash

- Prior to SMM_PFAT_PROTOCOL installation the following elements of PFAT Directory should be ready
    - PFAT platform data table (PPDT) – Sample code provided, SHA256 Hash of the PPDT has to passed to PFAT_CONFIG in CPU platform policy PPI. Hash can be either hardcoded value for given platform configuration/calculated in run time using libraries, if available in platform BIOS.
    - PUP Header – Sample code provided. Most of the information in PUP header is fixed for a given platform. PFAT module reads header and identifies if flash update is targeted to EC Flash/BIOS.
- For PFAT Flash Components are always treated as one composite flash component F0 irrespective of the physical flash components on platform. Address passed to PFAT services to write/erase to flash is offset address from base of composite flash component.
- Size allocated for PFAT module should always be 256K irrespective to size of PFAT module to eliminate AC-RAM overlap of PFAT directory elements. Please refer to PFAT EAS for more information on this.

3.6.4 Variable Writes with PFAT

All Non-volatile UEFI variable writes/erases should be done via PFAT Module and so has to be thru SMI Handler. SPI library[6] instances has to be modified to map to PFAT SMM protocol instead of PCH SPI protocol for writes/erases to flash. Rest of the flash operations will continue to use PCH SPI protocol.

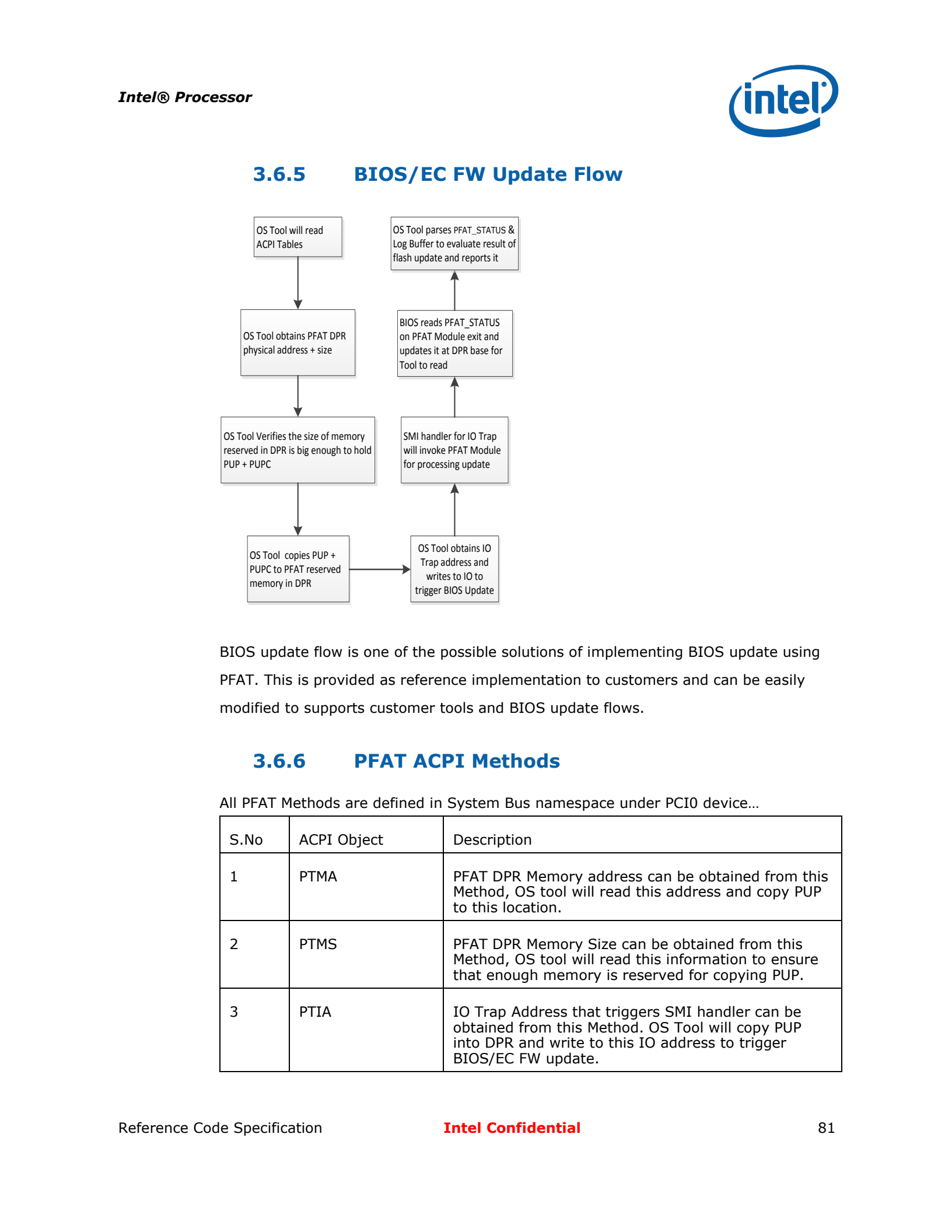
The fact that all flash writes/erases happen in SMM only any variable writes/erases can happen only after SMM Base is initialized and PFAT SMM protocol is installed. Also variable writes/erases that happen in DXE has to be proxied to SMM.

---

[6] Customers are responsible for modifying their SPI libraries & Variables writes to SMM & PFAT protocol. This is not provided as part of RC.

Eg: SPI Flash Write

```
EFI_STATUS
SpiFlashWrite (
  IN     UINTN                  Address,
  IN OUT UINT32                 *NumBytes,
  IN     UINT8                  *Buffer
  )
{
...
#ifdef PFAT_FLAG
  if (mPfatProtocol != NULL) {
    mPfatProtocol->Write (
                  mPfatProtocol,
                  (UINT32) Address,
                  RemainingBytes,
                  Buffer
                  );
    return EFI_SUCCESS;
  }
#endif
...
}
```



3.6.5 BIOS/EC FW Update Flow

BIOS update flow is one of the possible solutions of implementing BIOS update using PFAT. This is provided as reference implementation to customers and can be easily modified to supports customer tools and BIOS update flows.

3.6.6 PFAT ACPI Methods

All PFAT Methods are defined in System Bus namespace under PCI0 device…

S.No	ACPI Object	Description
1	PTMA	PFAT DPR Memory address can be obtained from this Method, OS tool will read this address and copy PUP to this location.
2	PTMS	PFAT DPR Memory Size can be obtained from this Method, OS tool will read this information to ensure that enough memory is reserved for copying PUP.
3	PTIA	IO Trap Address that triggers SMI handler can be obtained from this Method. OS Tool will copy PUP into DPR and write to this IO address to trigger BIOS/EC FW update.

3.6.7 Code Definitions

3.6.7.1 Depedencies

- ***EFI_SMM_BASE_PROTOCOL*** - Documented in *System Management Mode Core Interface Specification* (SmmCis.pdf)

3.6.7.2 PFAT Module

Signed PFAT module is delivered by Intel Corporation as separate binary. PFAT module is launched on all flash writes & erases

PFAT_MODULE_GUID

Summary

PFAT Services during SMM initialization use this GUID to locate PFAT module, and update PFAT Directory with its location.

GUID

```
#define PFAT_MODULE_GUID  \
  { 0x7934156D, 0xCFCE, 0x460E, 0x92, 0xF5, 0xA0, 0x79, 0x09, 0xA5, 0x9E, 0xCA}
```

3.6.7.3 PFAT HOB

Description

PFAT HOB is used for passing platform policy settings and CPU/chipset information to PFAT driver in SMM & DXE phase.

Definitions

#define PFAT_HOB_GUID \

{0x66F0C42D,0x0D0E,0x4C23,0x93,0XC0,0x2D,0x52,0x95,0xDC,0x5E,0x21}

typedef struct {

```
  EFI_HOB_GUID_TYPE     EfiHobGuidType;
  PPDT                  Ppdt;
  PUP_HEADER            PupHeader;
  UINT8                 NumSpiComponents;
  UINT8                 ComponentSize[MAX_SPI_COMPONENTS];
  UINT64                PfatToolsIntIoTrapAdd;
  PFAT_LOG              PfatLog;
} PFAT_HOB;
```

Parameters

EfiHobGuidType	
Ppdt	PFAT Platform Data Table, refer to PFAT EAS for more information on PPDT
PupHeader	PFAT update package header, all flash updates are appended to this header along with PSL script
NumSpiComponents	Number of physical SPI flash components on platform
ComponentSize	Array containing size of each flash component
UINT64	IO Trap address required to Initialize PFAT Tools Interface
PfatLog	Header for PFAT Log Buffer

3.6.7.4 SMM PFAT Protocol

PFAT Module can only be launched from SMM, this means that all flash writes & erases that BIOS needs to do must flow thru SMI Handler and so dependency on SMM_BASE_PROTOCOL for installing PFAT Protocol. Prior to PFAT SMM Protocol being installed there should be no writes/erases to flash.

Summary

This protocol provides all the services required for flash writes/erases via PFAT

GUID

```
#define SMM_PFAT_PROTOCOL_GUID  \
  { 0xc3e156e4, 0x27b3, 0x4dff, 0xb8, 0x96, 0xfb, 0x11, 0x3b, 0x2e, 0x68,
0xb5 }
```

Prototype

```
typedef struct _PFAT_PROTOCOL {
  PFAT_WRITE   Write;
  PFAT_ERASE   Erase;
  PFAT_EXECUTE Execute;
} PFAT_PROTOCOL;
```

Parameters

PFAT_WRITE

Invoked to fill up PFAT script buffer for flash writes

PFAT_ERASE

Invoked to fill up PFAT script buffer for flash erases

PFAT_EXECUTE

Will trigger invocation of PFAT module

3.6.7.5 PFAT_PROTOCOL.Write()

Summary

This service fills PFAT script buffer for flash writes.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *PFAT_WRITE) (
  IN     PFAT_PROTOCOL  * This,
  IN     UINTN          Address,
  IN     UINT32         DataByteCount,
  IN OUT UINT8          * Buffer
  );
```

Parameters

This

Protocol entry

Address

Offset Address to write to from start of BIOS region in flash

DataByteCount

Number of Bytes to write to flash

Buffer

Pointer to the buffer containing data to be written to flash

Description

BIOS should invoke this function prior to calling PFAT_PROTOCOL.Execute() with all the relevant data required for flash write. This function will not invoke PFAT Module, only create script required for writing to flash.

Related Definitions

None

Status Codes Returned

EFI_SUCCESS	Successfully created script required for writing to flash

3.6.7.6 PFAT_PROTOCOL.Erase()

Summary

This service fills PFAT script buffer for erasing block in flash.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *PFAT_ERASE) (
  IN     PFAT_PROTOCOL  * This,
  IN     UINTN          Address,
  );
```

Parameters

This

Protocol entry

Address

Address of the block that is going to be erased

Description

BIOS should invoke this function prior to calling PFAT_PROTOCOL.Execute() with all the relevant data required for flash erase. This function will not invoke PFAT module, only create script required for erasing each block in the flash

Related Definitions

None

Status Codes Returned

EFI_SUCCESS	Successfully created script required for erasing block in flash

3.6.7.7 PFAT_PROTOCOL.Execute()

Summary

This service will write PFAT_DIRECTORY MSR and invoke the PFAT Module by writing to PLAT_FRMW_PROT_TRIGGER MSR for writing/erasing to flash.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *PFAT_EXECUTE) (
  IN     PFAT_PROTOCOL  * This,
  IN     BOOLEAN        BiosUpdate
  );
```

Parameters

This

Protocol entry

BiosUpdate

Flag that indicates that a BIOS/EC FW update is requested

Description

BIOS should invoke PFAT_PROTOCOL.Write/Erase() function prior to calling PFAT_PROTOCOL.Execute() for flash writes/erases(except for BiosUpdate). Write/Erase() function will render PFAT script during execution. Execute() function will implement the following steps…

1. Update PFAT directory with address of PUP
2. All the AP's except the master thread are put to sleep.
3. PFAT module is invoked from BSP for completing flash operation.

If BiosUpdate flag is set to true, PUP (PUP Header + PFAT Script + Update data) is part of data that is passed to SMI Handler. SMI Handler invokes PFAT module to process the update.

Related Definitions

None

Status Codes Returned

EFI_SUCCESS	Successfully completed flash operation

3.7 Overclocking

3.7.1 Overclocking Overview

Significant changes have been made on Haswell in order to provide a more consolidated interface for overclocking. Part of these changes was required as a result of adding the integrated voltage regulator (iVR) and removing external voltage rails from the platform.

At a high level, clock ratio-based overclocking is supported on the IA core, GT, cache, and Uncore domains. Support includes voltage/frequency curve control for each of

these domains, maximum overclocking ratio overrides and input voltage regulator management.

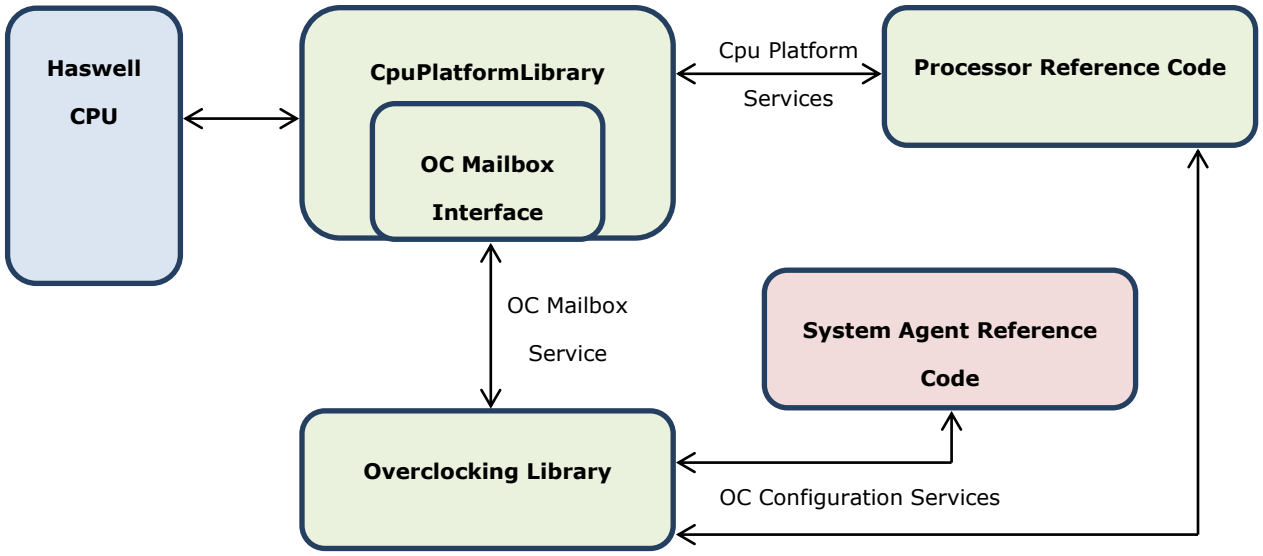
Voltage overrides are supported for several domains. At a high level, support includes:

- Direct voltage override across the entire range of operation
- Positive and negative offset across the entire range of operation
- Voltage "Interpolation" applied only to the overclocked core frequencies

Access to these overclocking controls are provided through the Overclocking Mailbox. For more detailed overclocking information refer to the Shark Bay Client Performance Tuning Guide.

3.7.2 Software Architecture

Configuration of most overclocking settings uses the services provided by two different libraries: CpuPlatform Library and the Overclocking Library. The Cpu Platform Library contains generic mailbox services while the Overclocking Library provides abstracted API's specific to overclocking. These libraries are also used for overclocking within the System Agent Reference Code. The following diagram shows the relationship between these libraries.



3.7.3 Overclocking Mailbox

Almost all overclocking controls will be managed through a new OVERCLOCKING_MAILBOX MSR (0x150). This MSR mailbox is new on Haswell and is a portal into PCU control and telemetry. This MSR based mailbox is implemented as a library in the processor reference code. Mailbox read and writes services are provided as part of the CPU platform library. These mailbox services can be accessed in both PEI and DXE.

3.7.4 Interfaces and Functions

This section describes mailbox interfaces provided by the Cpu platform library.

3.7.4.1 MailboxRead()

Summary

This procedure is a generic read of a mailbox interface.

Prototype

```
EFI_STATUS
MailboxRead (
  IN UINT32       MailboxType,
  IN UINT32       MailboxCommand,
  OUT UINT32      *MailboxDataPtr,
  OUT UINT32      *MailboxStatus
);
```

Parameters

MailboxType

The type of mailbox interface to read. The Overclocking mailbox is defined as MAILBOX_TYPE_OC = 2.

MailboxCommand

Overclocking mailbox command data

**MailboxDataPtr*

Pointer to the overclocking mailbox interface data

**MailboxStatus*

Pointer to the mailbox status returned from pcode. Possible mailbox status values are:

SUCCESS (0)	Command succeeded.
OC_LOCKED (1)	Overclocking is locked. Service is read-only.
INVALID_DOMAIN (2)	Invalid Domain ID provided in command data.
MAX_RATIO_EXCEEDED (3)	Ratio exceeds maximum overclocking limits.
MAX_VOLTAGE_EXCEEDED (4)	Voltage exceeds input VR's max voltage.
OC_NOT_SUPPORTED (5)	Domain does not support overclocking.

Description

This procedure performs a read request from the mailbox type provided. The return data is copied to the MailboxDataPtr.

Status Codes Returned

EFI_SUCCESS	Command succeeded.

EFI_INVALID_PARAMETER	Invalid read data detected from pcode.
EFI_UNSUPPORTED	Unsupported MailboxType parameter.

3.7.4.2 MailboxWrite()

Summary

This procedure is a generic write to a mailbox interface.

Prototype

```
EFI_STATUS
MailboxRead (
  IN UINT32       MailboxType,
  IN UINT32       MailboxCommand,
  OUT UINT32      MailboxDataPtr,
  OUT UINT32      *MailboxStatus
);
```

Parameters

MailboxType

The type of mailbox interface to read. The Overclocking mailbox is defined as MAILBOX_TYPE_OC = 2.

MailboxCommand

Overclocking mailbox command data

MailboxDataPtr

Overclocking mailbox interface data

**MailboxStatus*

Pointer to the mailbox status returned from pcode. Possible mailbox status values are:

SUCCESS (0)	Command succeeded.
OC_LOCKED (1)	Overclocking is locked. Service is read-only.
INVALID_DOMAIN (2)	Invalid Domain ID provided in command data.

MAX_RATIO_EXCEEDED (3)	Ratio exceeds maximum overclocking limits.
MAX_VOLTAGE_EXCEEDED (4)	Voltage exceeds input VR's max voltage.
OC_NOT_SUPPORTED (5)	Domain does not support overclocking.

Description

This procedure performs a write to the mailbox type provided. The status of the write request is copied to MailboxStatus.

Status Codes Returned

EFI_SUCCESS	Command succeeded.
EFI_INVALID_PARAMETER	Invalid read data detected from pcode.
EFI_UNSUPPORTED	Unsupported MailboxType parameter.

For complete overclocking mailbox command and interface data definitions please refer to the Shark Bay Performance Tuning Guide.

3.7.5 Overclocking Library

In addition to the overclocking mailbox service there is a separate overclocking library included in the Processor Reference Code. The overclocking library provides an abstracted interface for CPU reference code components to interact with the OC mailbox MSR interface. The intent of this library is to provide an easy interface to configure overclocking settings. The overclocking library provides the following services:

- GetVoltageFrequencyItem()
    - Gets the overclocking voltage, frequency, and max ratio information from all of the CPU domains: IA core, GT, Cache, Uncore, IOA, and IOD.
- SetVoltageFrequencyItem()
    - Sets the overclocking voltage, frequency, and max ratio information for any of the CPU domains: IA core, GT, Cache, Uncore, IOA, and IOD.

- GetSvidConfig()
    - Gets the current external VR voltage level.
- SetSvidConfig()
    - Sets the external VR voltage level.
- GetOcCapabilities()
    - Gets the overclocking capabilities for all CPU domains: IA core, GT, Cache, Uncore, IOA, and IOD.
- GetFivrConfig()
    - Gets the iVR configuration information.
- SetFivrConfig()
    - Sets the iVR configuration information.

Detailed interface definitions can be found in the RcCpuApi help file.

3.8 HowTo

3.8.1 How processor code perform Microcode Update in POST and S3?

POST:

CpuInitDxe driver uses **DXE_CPU_POLICY_PROTOCOL. RetrieveMicrocode ()** interface to get the microcode location.

CpuInitDxe driver copies the Microcode into SMM region

CpuInitDxe driver perform MCU update via the flow in BWG by MpServices for all CPUs.

S3:

During S3 resume path, CpuS3Peim get the Microcode in SMM region.

CpuS3Peim uses its simple MpService to perform Microcode update for all CPUs.

3.8.2 How Remap is done in S3(PEIM) and POST(DXE)?

POST:

1. SmmBaseRuntime driver loads SmmRelocDxe to perform the remap.
2. SmmBaseRuntime driver loads SmmRelocPeim to SMM region with some information

S3:

1. SmmBasePeim driver execute SmmRelocPeim in S3 resume path to perform remap work.
2. BIOS follow the ACPI spec to return to OS wake vector

3.8.3 How to provide cache layout when memory is ready?

Sample Code 1: Notify by gPeiMemoryDiscoveredPpiGuid

```
  EFI_STATUS                    Status;
  EFI_PEI_HOB_POINTERS          Hob;
  PEI_CACHE_PPI                 *CachePpi;
  UINT64                        MemoryBase;
  UINT64                        MemoryLength;
  EFI_BOOT_MODE                 BootMode;

  //
  // Load Cache PPI
  //
  Status = (**PeiServices).LocatePpi (
                            PeiServices,
                            &gPeiCachePpiGuid,  // GUID
                            0,                  // Instance
                            NULL,               // PEI_PPI_DESCRIPTOR
                            &CachePpi           // PPI
                            );

  if (!EFI_ERROR (Status)) {
    Status = (**PeiServices).NotifyPpi (PeiServices, &mMtrrNotifyList);
    ASSERT_PEI_ERROR (PeiServices, Status);

    //
    // It will reset all MTRR setting.
    //
    CachePpi->ResetCache (
                PeiServices,
                CachePpi
                );
    //
    // Cache the Flash area as WP to boost performance
    //
    CachePpi->SetCache (
                PeiServices,
                CachePpi,
                FLASH_BASE,
                FLASH_SIZE,
                EFI_CACHE_WRITEPROTECTED
                );

    //
    // Assume size of main memory is multiple of 256MB
    //
    MemoryLength = (LowMemoryLength + 0xFFFFFFF) & 0xF0000000;
    MemoryBase = 0;
    CachePpi->SetCache (
```

```
                PeiServices,
                CachePpi,
                MemoryBase,
                MemoryLength,
                EFI_CACHE_WRITEBACK
                );

    MemoryBase = LowMemoryLength;
    MemoryLength -= LowMemoryLength;
    CachePpi->SetCache (
                PeiServices,
                CachePpi,
                MemoryBase,
                MemoryLength,
                EFI_CACHE_UNCACHEABLE
                );
  //
  // Disable NEM, Update MTRR setting from MTRR buffer
  //
  CachePpi->ActivateCache (PeiServices, CachePpi);
  }

  AsmCpuid (0x80000000, &RegEax, NULL, NULL, NULL);
  if (RegEax >= 0x80000008) {
    AsmCpuid (0x80000008, &RegEax, NULL, NULL, NULL);
    PhysicalAddressBits = (UINT8) RegEax;
  } else {
    PhysicalAddressBits = 36;
  }

  //
  // Create a CPU hand-off information
  //
  Status = PeiBuildHobCpu (PeiServices, PhysicalAddressBits, 16);
```

Sample code 2: Notify by gEndOfPeiSignalPpiGuid

```
  //
  // Load Cache PPI
  //
  Status = (**PeiServices).LocatePpi (
                            PeiServices,
                            &gPeiCachePpiGuid,  // GUID
                            0,                  // Instance
                            NULL,               // PEI_PPI_DESCRIPTOR
                            &CachePpi           // PPI
                            );

  if (!EFI_ERROR (Status)) {
    //
```

```
    // Clear the CAR Settings
    //
    Status = CachePpi->ResetCache (
                        PeiServices,
                        CachePpi
                        );
    ASSERT_PEI_ERROR (PeiServices, Status);

    //
    // Set fixed cache for memory range below 1MB
    //
    Status = CachePpi->SetCache (
                PeiServices,
                CachePpi,
                0x0,
                0xA0000,
                EFI_CACHE_WRITEBACK
                );
    ASSERT_PEI_ERROR (PeiServices, Status);

    Status = CachePpi->SetCache (
                PeiServices,
                CachePpi,
                0xA0000,
                0x20000,
                EFI_CACHE_UNCACHEABLE
                );
    ASSERT_PEI_ERROR (PeiServices, Status);

    Status = CachePpi->SetCache (
                PeiServices,
                CachePpi,
                0xC0000,
                0x40000,
                EFI_CACHE_WRITEPROTECTED
                );
    ASSERT_PEI_ERROR (PeiServices, Status);

...
    //
    // Set non system memory as UC
    //
    MemoryBase   = 0x100000000;

    //
    // Add IED size to set whole SMRAM as WB to save MTRR count
    //
    SmramSize += 0x400000;
    MemoryLength = MemoryBase - (SmramBase + SmramSize);
```

```
    while (MemoryLength != 0) {
      Power2Length = GetPowerOfTwo (MemoryLength);
      MemoryBase -= Power2Length;
      CachePpi->SetCache (
                  PeiServices,
                  CachePpi,
                  MemoryBase,
                  Power2Length,
                  EFI_CACHE_UNCACHEABLE
                  );

      MemoryLength -= Power2Length;
    }

    //
    // Update MTRR setting from MTRR buffer
    //
    CachePpi->ActivateCache (PeiServices, CachePpi);

    //
    // Default Cachable attribute will be set to WB to support large
memory size/hot plug memory
    //
    SetDefaultMemoryCacheAsWB ();
```

3.8.4 Responsiveness

TBD

4 Integration Guide

4.1 Integration Overview

The reference code needs to be properly integrated into a Framework source code base to be utilized effectively. It is assumed that the target code base utilizes an EDK from the TianoCore.org website. Failure to use the proper EDK code may result in the need to make significant modifications to the reference code.

Topics included in this guide are:

- Reference code package contents
- Integration checklist
- Building with EDK1117 Instructions
- Building with Patch 8 Instructions
- Required and recommended porting details for the sub modules.

It is expected that firmware developers wishing to support Framework implementations utilizing Intel Haswell hardware will use this specification as a guide for how to include, review, modify, and validate the *Framework Client Bios Haswell Processor Reference Code*.

4.2 Reference Code Package Contents

The Framework Client Bios Haswell Processor Reference code package contains:

- *Framework Client Bios Haswell Processor Reference Code Specification*
  - This document
- *Framework Client Bios Haswell Processor Reference Code*

  Framework-compliant code written per architecture and design requirements to support common features for the Shark Bay platform.

- *Release Note*

4.3 Integration Checklist

Please follow this checklist for integrating the reference code:

1. Integrate the following reference codes into the EDK source in advance:
   a. *Framework Client Bios System Agent Reference Code*
   b. *Framework Client Bios LynxPoint PCH Reference Code*
2. Unzip the reference code to a source directory that is commonly accessible to all platforms.
3. Review each module:
   a. Review module description.
   b. Verify prerequisites documented in the *Intel Haswell processor Reference Code Design Specification* are met.
   c. Verify Integration Steps documented in Section **Error! Reference source not found.** are completed.
4. Build and test each module's functionality.
5. Review release notes.

Note: Sample DSC files are provided for direct integration into platform DSC files. There are DSC files for the Libraries, and DXE or later modules.

If needs to support TxT, please follow below checklist.

1. Proper BIOS ACM file for the platform must be obtained through your Intel Representative and placed into Txt\BiosAcm folder.
2. STAFIXUP command needs to be added proper into build process.
3. To build TxtPeiAp.bin requires Link16.exe which is 32 bit link tool.
4. Take the necessary steps to include BIOSAC_xxxxx.FFS, TXTPEI.FFS and TXTPEIAP.BIN into the boot block FV and must be located at the highest 128KB of BIOS image.
5. MemoryInit PEIM of SA reference code needs to set TXT_SUPPORT_FLAG for enabling TxT support. It is required to ensure the flag is set for MemoryInit if TxT code is integrated.

4.4 Building with EDK1117 Instructions

The reference code modules are designed to be built with the EDK or EDK derived products. Building with the EDK may be useful to baseline build results as different core code bases may vary widely in build infrastructure or environment.

Note: Please get EDK V8 Patch from tianocore.org.

Here are the steps to build the reference code with the EDK20081117 (V8 applied) environment:

1. Integrate PCH and SA reference code first.
2. Copy Processor Reference Code to $(EDK_SOURCE)\Edk\Sample\Cpu\Haswell
3. Modify Common.dsc under Sample\Platform:

   From:

   ```
   [Compile.Ia32.asm,Compile.x64.asm]
   ```

   To:

   ```
   [Compile.Ia32.asm16,Compile.Ia32.asm,Compile.x64.asm]
   ```

4. Change PlatformTools.env under x64\Build, add these macros:

```
PROJECT_NAME = x64
PROJECT_CPU_ROOT = Sample\Cpu\Haswell
```

5. Change the Makefile in x64\Build folder, add these lines to the ProcessDsc calls:

```
-d PROJECT_CPU_ROOT=$(PROJECT_CPU_ROOT)\
```

6. Modify the following line in the `[Libraries.Platform]` section after `DEFINE PROCESSOR=IA32` on X64.dsc:

   From:

```
!include "$(EDK_SOURCE)\Sample\Platform\EdkLib32.dsc"
```

To:

```
!include "$(EDK_SOURCE)\Sample\Platform\EdkLibAll.dsc"
```

7. Add EdkII Glue Library, MePeiLib in the `[Libraries.Platform]` section after `DEFINE PROCESSOR=IA32` on X64.dsc:

```
!include "$(EDK_SOURCE)\Sample\Platform\EdkIIGlueLib32.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_CPU_ROOT)\Include\IntelCpuPeiLib.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_PCH_ROOT)\Include\IntelPchPeiLib.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_SA_ROOT)\Include\IntelSaPeiLib.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_ME_ROOT)\Include\MePeiLib.dsc"
```

8. Add EdkII Glue Library and Intel processor Library in the `[Libraries.Platform]` section after `DEFINE PROCESSOR=X64` on X64.dsc:

```
!include "$(EDK_SOURCE)\Sample\Platform\EdkIIGlueLibAll.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_CPU_ROOT)\Include\IntelCpuDxeLib.dsc"
!include
"$(EDK_SOURCE)\$(PROJECT_CPU_ROOT)\SampleCode\Include\IntelCpuDxeSampleCo
de.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_PCH_ROOT)\Include\IntelPchDxeLib.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_SA_ROOT)\Include\IntelSaDxeLib.dsc"
!include "$(EDK_SOURCE)\$(PROJECT_ME_ROOT)\Include\MeDxeLib.dsc"
```

9. Add DSC build extensions in the x64build folder (Edk\Sample\Platform\x64\Build) x64.dsc, in the [Defines] section to build ACPI tables and PPM code,

```
!include
"$(EFI_SOURCE)\$(PROJECT_CPU_ROOT)\SampleCode\Include\AcpiBuild.
dsc"
!include
”$(EFI_SOURCE)\$(PROJECT_CPU_ROOT)\Include\CpuPowerMgmt.dsc
```

10. Add the Pei modules individually or add following line in the `[Components]` section after `DEFINE FV=Fv DEFINE PROCESSOR=IA32` on X64.dsc

```
!include "$(EDK_SOURCE)\$(PROJECT_CPU_ROOT)\Include\IntelCpuPei.dsc"
```

Add the Dxe modules individually or the following line in the `[Components]` section after `DEFINE FV=Fv DEFINE PROCESSOR=X64` on X64.dsc

```
!include "$(EDK_SOURCE)\$(PROJECT_CPU_ROOT)\Include\IntelCpuDxe.dsc"
```

11. Define EDK build configuration on Sample\Platform\X64\Build\Config.env

```
EFI_S3_RESUME = YES
```

12. If you are building with Visual Studio 2008, modify Sample\Platform\X64\Build\X64.dsc to comment out these components or resolve the build issues generated (Some sample resolutions available in EDK 1.05 or later)

```
#Sample\Universal\Ebc\Dxe\Ebc.inf
#Sample\Universal\UserInterface\HiiDataBase\Dxe\HiiDatabase.inf
#Sample\Bus\Pci\PciBusNoEnumeration\Dxe\PciBusNoEnumeration.inf
#Sample\Universal\UserInterface\SetupBrowser\Dxe\SetupBrowser.Inf
Run nmake in $(EDK_SOURCE)\Sample\Platform\x64\build directory.
```

Refer to the *EFI Developer Kit (EDK) Getting Started Guide* for other EDK build instructions.

It is not guaranteed that modules built with the EDK are valid for use with real hardware. Building with EDK provides guidance for integrating reference code into any Framework-based system BIOS and is meant for reference purposes only.

4.5 MASM

Haswell CPU reference code contains 16/32 bit assembly code. It requires 16/32 bit assembler and linker. Please ensure both 16/32 bit assembler and linker are installed. Assume 16/32 bit linker is Link16.exe. Follow below instructions.

1. Copy Link16.exe to $(MASMPATH)\binr. $(MASMPATH) is defined in '\Sample\LocalTools.env' as 'c:\masm611'

2. Modify Sample\CommonTools.env with below lines following

   For IA32 build, it requires Link16.exe to build 16/32 bit assembly code.
   !IF "$(EFI_ASSEMBLER_NAME)" == ""

   From:

ASKLINK =$(MASMPATH)\binr\link16

To:

ASMLINK =”$(VCINSTALLDIR)\bin\link”

For X64 build,

!IF “$(EFI_ASSEMBLER_NAME)” == “”

From:

ASM =$(MASMPATH)\bin\ml

ASKLINK =$(MASMPATH)\binr\link

To:

ASM =”$(VCINSTALLDIR)\bin\ml”

ASMLINK =”$(VCINSTALLDIR)\bin\link”

4.6 Visual Studio 2008 SP1 Support

The modules are designed to be built with the EDK source by Visual Studio 2008 SP1. If you are using Visual Studio 2008 please follow below additional instructions.

1. Modify all necessary files basing on 2.4 and integration guides, including H, DSC, INF, ENV and Makefile…etc.

2. Modify Sample\CommonTools.env with below lines following !IF "$(EFI_COMPILER_X64_NAME)"==""

   From:

   ```
   CC          =$(WIN_DDK_X64_PATH)\cl
   LINK        =$(WIN_DDK_X64_PATH)\link
   LIB         =$(WIN_DDK_X64_PATH)\lib
   ASM         =$(WIN_DDK_X64_PATH)\ml64
   ```

   To:

   ```
   CC          ="$(VCINSTALLDIR)\bin\x86_amd64\cl"
   LINK        ="$(VCINSTALLDIR)\bin\x86_amd64\link"
   LIB         ="$(VCINSTALLDIR)\bin\x86_amd64\lib"
   ASM         ="$(VCINSTALLDIR)\bin\x86_amd64\ml64"
   ```

3. Define your EDK build configuration on Sample\Platform\x64\Build\Config.env

   ```
   UEFI_MODE = YES
   USE_VC8 = YES
   ```

4. Modify Sample\Platform\x64\Build\x64.dsc to comment out below components or resolve the build issues they generated (Some sample resolutions available in EDK 1.05 or later)

   ```
   #Sample\Universal\Ebc\Dxe\Ebc.inf
   #Sample\Universal\UserInterface\HiiDataBase\Dxe\HiiDatabase.inf
   #Sample\Bus\Pci\PciBusNoEnumeration\Dxe\PciBusNoEnumeration.inf
   #Sample\Universal\KUserInterface\SetupBrowser\Dxe\SetupBrowser.inf
   ```

Call vsvar32.bat from your local Visual Studio path (ex: Call "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat") and set build tip to Sample\Platform\x64\build. Following normal build process to build the source.

4.7 Platform Configuration Requirements

4.7.1 Overview

The reference code does not completely configure all aspects of the Haswell Processor hardware. These tasks may be required and are not currently performed by the reference code:

- Initialize CPU Platform Policy PPI, which is consumed by CpuInit PEIM and other PEI phase modules.
- Initialize CPU Platform Policy Protocol, which is consumed by CpuInit DXE driver and other DXE drivers.

4.7.2 CPU Platform Policy PPI Initialization

The user needs to implement the CPU PLATFORM POLICY PPI in a platform module. CpuInit PEI and other PEI Modules require the PPI to initialize Processor in PEI phase.

```
struct _PEI_CPU_PLATFORM_POLICY_PPI {
  UINT8                  Revision;

  CPU_CONFIG_PPI             *CpuConfig;
  POWER_MGMT_CONFIG_PPI      *PowerMgmtConfig;
  SECURITY_CONFIG_PPI        *SecurityConfig;
  OVERCLOCKING_CONFIG_PPI      *OverclockingConfig;

};
```

Sample Code

Please refer to SampleCode\CpuPolicyInit\Pei for Policy defaults.

Suggested Location

The PPI has to be published before CpuInit PEIM are dispatched.

4.7.3 CPU Platform Policy Protocol Initialization

The platform code must publish the SA Platform Policy Protocol to pass platform settings to SaInit, PciExpress and SmbiosMemory Dxe drivers.

```
typedef struct _DXE_CPU_PLATFORM_POLICY_PROTOCOL {
  UINT8                         Revision;

  CPU_CONFIG                    *CpuConfig;
  POWER_MGMT_CONFIG             *PowerMgmtConfig;
  SECURITY_CONFIG               *SecurityConfig;

} DXE_CPU_PLATFORM_POLICY_PROTOCOL;
```

Sample Code

Please refer to SampleCode\CpuPolicyInit\Dxe for Policy defaults.

Suggested Location

The protocol has to be installed before CpuInitDxe, PowerMgmtInit, TxtDxe drivers are dispatched.

4.7.4 Build Flags

CPU reference code defines below build flags for enable/disable function support.

BOOT_GUARD_SUPPORT_FLAG – Set the flags to 1 for enable Boot Guard support

4.7.5 Definition changes highlight

In CPU RC v1.3.0, there are two definitions “B_TPM_DISCONNECT” and “B_BOOT_GUARD_ENF_MASK” are from ME RC MeChipset.h.