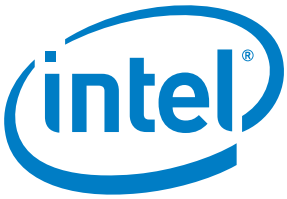


White Paper

Open Braswell UEFI Codebase

Design and Porting Guide

Feb 2016

David Wei
Mike Wu
Vincent Zimmer
Chris Li

<DOC NUMBER>

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the architecture of UEFI codebase of Open Braswell, how to integrate Intel® Firmware Support Package (FSP) into Open Braswell codebase, how to port Open Braswell codebase on custom Braswell-SoC based mainboard. The UEFI BIOS architecture designed by this document also could be applied on other Intel platform.

1.2 Intended Audience

This document is targeted at all platform and system developers who need to customize Open Braswell codebase for Braswell-SoC based mainboard. This includes, but is not limited to: system BIOS developers, system integrators, as well as Intel® Firmware Support Package (FSP) provider.

1.3 Prerequisite

- This paper assumes that audience has EDKII/UEFI firmware development experience. He or she should be familiar with *Platform Initialization (PI) Specification* V1.4 and *UEFI Specification V2.5.*
- Open Braswell EDKII codebase comply with *Intel® Firmware Support Package:* External Architecture Specification V1.1 and assumes that audience has full understanding of *Intel® Firmware Support Package:* External Architecture Specification V1.1.
- Open Braswell is based on UEFI development Kit (UDK) 2015 (UDK2015) Release of EDKII project.

1.4 Related Documents and Tools

- *Platform Initialization (PI) Specification* located at http://www.uefi.org/specifications
- *UEFI Specification* located at http://www.uefi.org/specifications
- *Intel® Firmware Support Package:* External Architecture Specification V1.1

  http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/fsp-architecture-spec.pdf
- *Binary Configuration Tool for Intel® Firmware Support Package* – available at http://www.intel.com/content/www/us/en/embedded/software/fsp/binary-configuration-tool-windows-download-tool.html
- *A Tour Beyond BIOS Using the Intel® Firmware Support Packagewith the EFI Developer Kit II*

http://firmware.intel.com/share/

- *Advanced Configuration and Power Interface Specification Revision 5.0*
  http://www.acpi.info

1.5 Acronyms and Terminology

Table 1. Acronyms and Terminology

Acronym	Definition
BCT	Binary Configuration Tool
BSF	Boot Setting File
BWG	BIOS Writer's Guide
CRB	Customer Reference Board
FSP	Firmware Support Package
UPD	Updatable Product Data
VPD	Vital Product Data

§

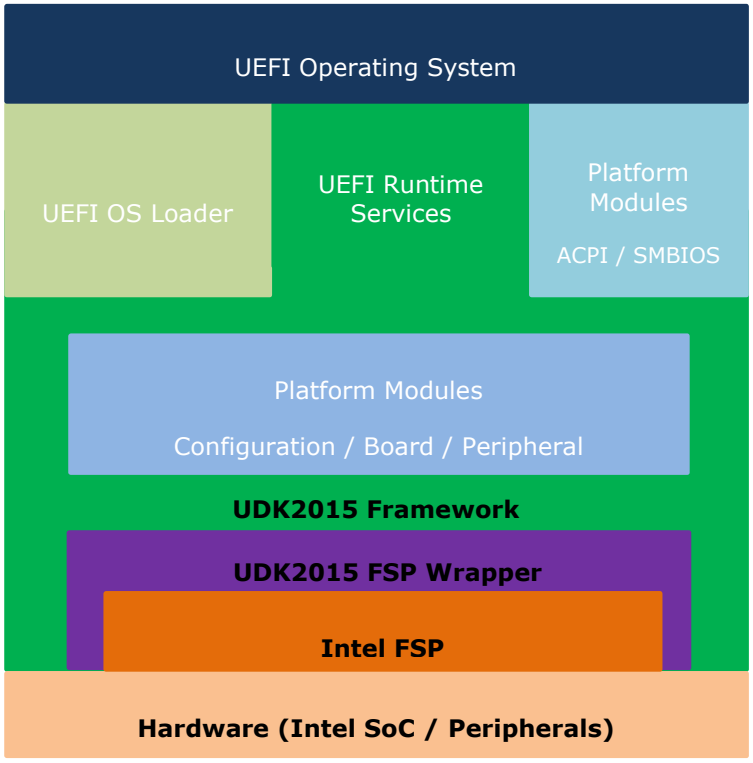
2 Open Braswell UEFI BIOS Design Guide

2.1 High Level Architecture of Open Braswell UEFI BIOS

Open Braswell UEFI BIOS consists of four major components, which are

- UEFI Development Kit (UDK2015) Release
- Intel Braswell Firmware Support Package (FSP)
- UDK2015 FSP Wrapper Packages
- Braswell Platform Packages

Below diagram shows the high level architecture of UEFI and FSP compatible BIOS for Braswell. Following sections of this chapter will introduce above four components in detail.



2.2 UEFI Development Kit 2015 (UDK2015) Release

Open Braswell is based on EDKII's UDK2015 release. UDK2015 provides drivers that are platform or silicon independent, which builds up the framework of the whole UEFI compliant codebase.

Open Braswell UEFI BIOS uses following UDK2015 packages, which could be get from http://www.tianocore.org/udk/udk2015/ .

- `MdePkg`
- `MdeModulePkg`
- `IntelFrameworkPkg`
- `IntelFrameworkModulePkg`
- `FatBinPkg`
- `PcAtChipsetPkg`
- `PerformancePkg`
- `UefiCpuPkg`
- `ShellBinPkg`
- `SecurityPkg`
- `CryptoPkg`

2.3 Intel® Firmware Support Package (FSP) for Braswell

The Intel® Firmware Support Package (FSP) provides basic Intel chipset and processor initialization which is classified as Intel confidential. It is shipped in a format that can easily be incorporated into many existing boot loaders.

FSP of Intel® Pentium® and Celeron® Processor N3000 Product Families and Intel® Atom™ x5-Z8300 Processor (formerly Braswell, Compliant with FSP v1.1 Specification) could be downloaded from `http://www.intel.com/fsp`.

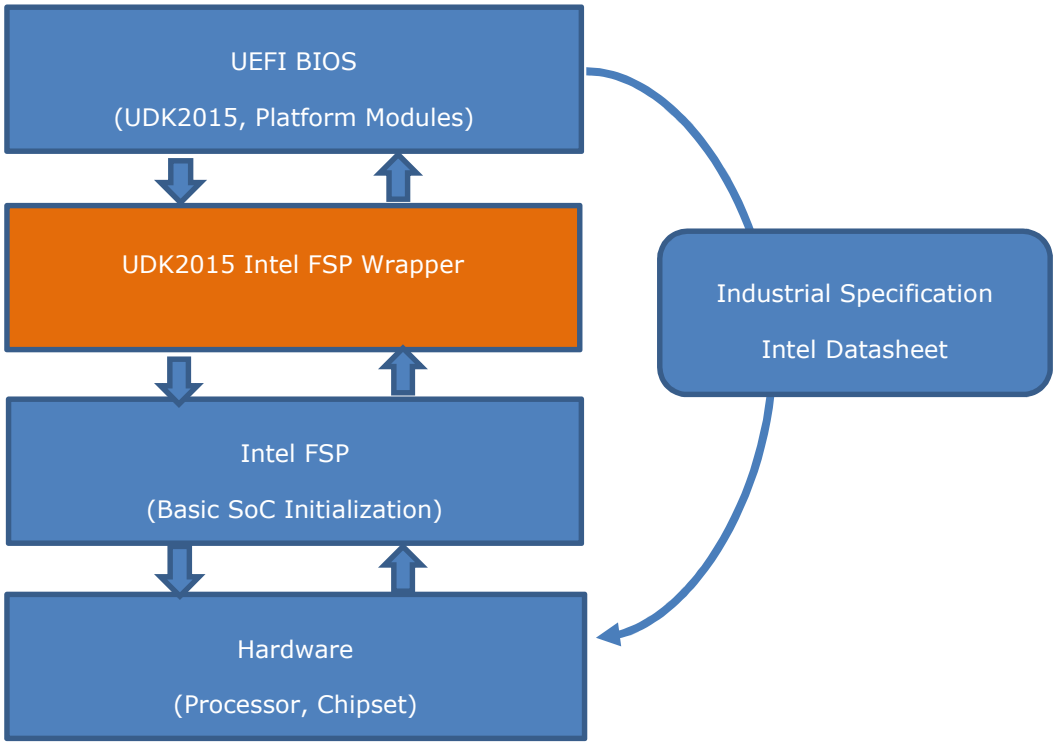
The Braswell FSP distribution package contains the following:

- FSP Binary and Boot Setting File(BSF)
- VPD/UPD Data structure definitions
- Braswell FSP Integration Guide
- Graphics Video BIOS Table (VBT) files and Graphics VBT BSF file.

2.4 UDK2015 FSP Wrapper Packages

The UDK2015 FSP Wrapper is part of UDK2015 release. It setups the operating environment for the FSP, provides thunk interface for UEFI BIOS to communicate with FSP API, calls FSP API in proper sequence and at proper timing to initialize Intel silicon. Following diagram xx illustrates the position of FSP wrapper in the system.

Note: UDK2015 and Platform Modules directly access interfaces which follow industrial specification or are published by Intel datasheet.



Open Braswell UEFI codebase includes following UDK2015 FSP wrapper packages, which could be get from http://www.tianocore.org/udk/udk2015/.

- `IntelFspPkg`
- `IntelFspWrapperPkg`

2.5 Braswell Platform Packages

Braswell platform packages provides platform specific modules, which cannot be reused by different platforms or different boards of the same platform.

Following two platform package are available at https://ssvn.intel.com/ssg/csd/tiano/tianoad/trunk/OC/Firmware/Braswell/ (TODO: Will be replaced by open source link)

- `BraswellPlatformPkg`
- `ChvRefCodePkg`

ChvRefCodePkg modules provides following four types of services:

- DXE architectural protocols
- UEFI Services
- Storage (SPI, SATA, SD/eMMC) related protocols
- System Management Mode foundation and SMM dispatch protocol

- Multi-Processor and Processor Power Management
- ACPI table / SMBIOS table

Module Name	Produced Interfaces
PowerManagement	ACPI SSDT of Processor C-state, P-state and T-state
SmmAccess	EFI_SMM_ACCESS2_PROTOCOL
PchSmiDispatcher	EFI_SMM_USB_DISPATCH2_PROTOCOL EFI_SMM_SX_DISPATCH2_PROTOCOL EFI_SMM_SW_DISPATCH2_PROTOCOL EFI_SMM_GPI_DISPATCH2_PROTOCOL EFI_SMM_ICHN_DISPATCH_PROTOCOL EFI_SMM_POWER_BUTTON_DISPATCH2_PROTOCOL EFI_SMM_PERIODIC_TIMER_DISPATCH2_PROTOCOL
Reset	EFI_RESET_ARCH_PROTOCOL RuntimeServices ResetSystem()
SataController	EFI_IDE_CONTROLLER_INIT_PROTOCOL
SDControllerDxe	EFI_SD_HOST_IO_PROTOCOL
SDMediaDeviceDxe	EFI_BLOCK_IO_PROTOCOL
SmmControl	EFI_SMM_CONTROL2_PROTOCOL
SPI (RuntimeDxe)	EFI_SPI_PROTOCOL
SPI (SMM)	EFI_SMM_SPI_PROTOCOL
CpuInit	EFI_CPU_ARCH_PROTOCOL EFI_MP_SERVICES_PROTOCOL
PiSmmCpuDxeSmm	EFI_SMM_CONFIGURATION_PROTOCOL EFI_SMM_CPU_PROTOCOL EFI_SMM_CPU_SYNC_PROTOCOL EFI_SMM_CPU_SYNC2_PROTOCOL EFI_SMM_CPU_SERVICE_PROTOCOL EFI_SMM_CPU_SAVE_STATE_PROTOCOL

BraswellPlatformPkg modules provides following services:

- Platform meta-data files (DSC, FDF)
- Platform library instance of UDK2015 defined platform library class
- Platform configuration (EDKII Setup, Platform Policy Protocol)
- SMI Handler
- Onboard Peripheral devices (Flash)
- Onchip PCU (Platform Control Unit) devices (UART, GPIO)

Note: Chapter 4 will give details on how to do platform porting.

3 Braswell FSP Integration Guide

The Intel® Firmware Support Package (FSP) provides chipset and processor initialization code in binary format that can easily be incorporated into UEFI BIOS.

Below are some steps for integrating FSP into UEFI BIOS:

- **Customizing (Optional)**

  Updating FSP binary's VPD and UPD data region with Intel Binary Configuration Tool (BCT) for different platform configuration.

- **Rebasing (Optional)**

  Rebasing FSP binary with Intel Binary Configuration Tool (BCT) if FSP will not be placed in the preferred location in UEFI BIOS image.

- **Placing**

  Placing FSP binary into the specific rebased location in UEFI BIOS image by modifying UEFI BIOS platform FDF file.

- **Interfacing**

  Adding UDK2015 FSP Wrapper into UEFI BIOS, which will setups the operating environment for the FSP, call the FSP with the correct parameters and parse the FSP output to retrieve the necessary information returned by the FSP.

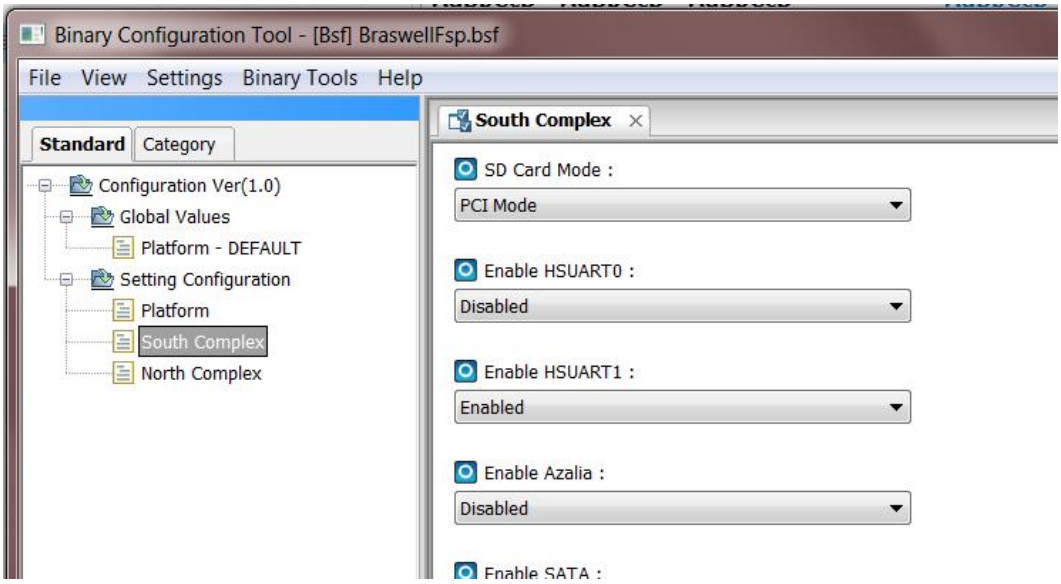
3.1 Customizing

The FSP binary contains a configurable data region, named VPD and UPD data region, which will be used by the FSP during the initialization.

Note: Please refer Chapter 8 in the FSP External Architecture Specification version 1.1 for details of VPD and UPD.

In order for FSP consumer to configure FSP binary's VPD and UPD region, a BSF (Binary Setting File) file is released with the FSP distribution package. Intel BCT (Binary Configuration Tool) opens the BSF file and shows configuration options to FSP consumer. FSP consumer could change those configuration options and patch FSP binary.

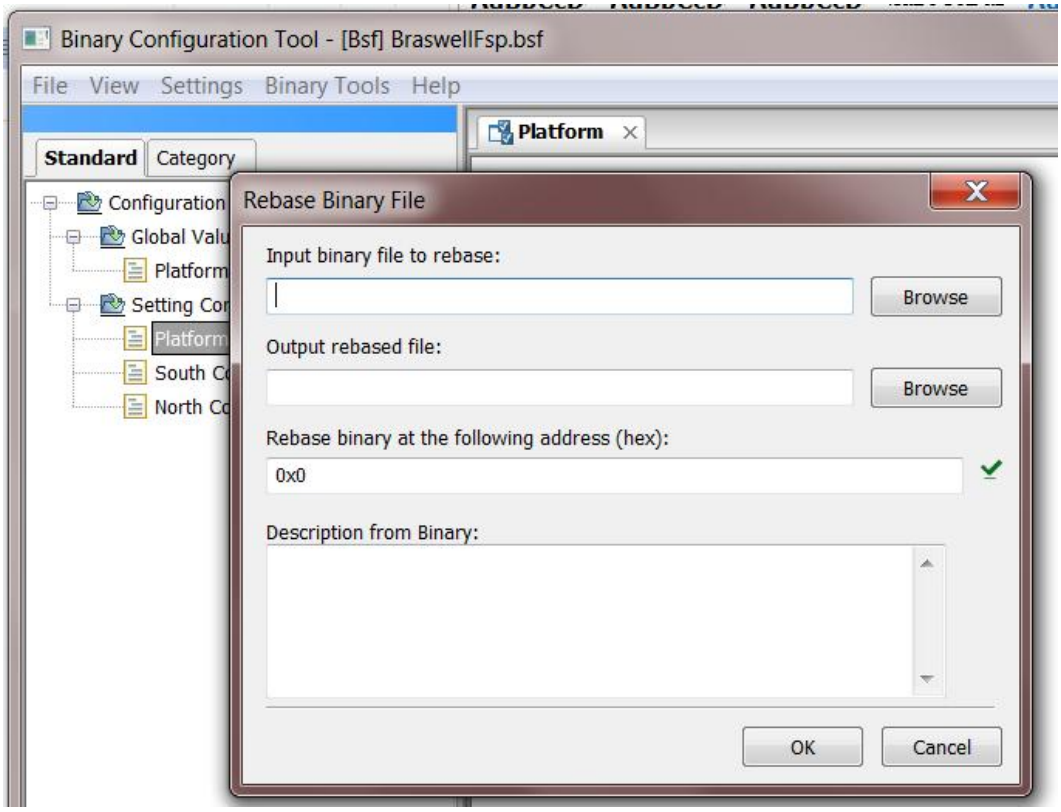
Note: Intel BCT tools is available at
http://www.intel.com/content/www/us/en/embedded/software/fsp/binary-configuration-tool-windows-download-tool.html



3.2 Rebasing

The FSP is not Position Independent Code (PIC) and the whole FSP has to be rebased if it is placed at a location which is different from the preferred address.

The FSP for the Braswell platform is built with a preferred base address of 0xFFF200000~~0xFFF6F000~~ according to Integration Guide, so the reference code provided in FSP assumes that the FSP is placed at this base address during the final UEFI BIOS build. FSP binary could be rebased to a different location with Intel Binary Configuration Tool (BCT).



3.3 Placing

FSP binary is placed into UEFI BIOS image as a Firmware Volume (FV). The base address of FSP FV should be either the preferred address documented in FSP Integration Guide, or a rebased address by BCT tool.

Below sample FDF meta-data of FSP placing is from Open Braswell platform FDF file.

[FD.Cht]

BaseAddress = $(FLASH_BASE) | gPlatformModuleTokenSpaceGuid.PcdBiosImageBase

Size = $(FLASH_SIZE) | gPlatformModuleTokenSpaceGuid.PcdBiosImageSize

…

$(FLASH_REGION_FSP_OFFSET) | $(FLASH_REGION_FSP_SIZE)

gFspWrapperTokenSpaceGuid.PcdFlashFvFspBase |
gFspWrapperTokenSpaceGuid.PcdFlashFvFspSize

FILE = ChvFspBinPkg/FspBinary/BSWFSP.fd

3.4 Interfacing

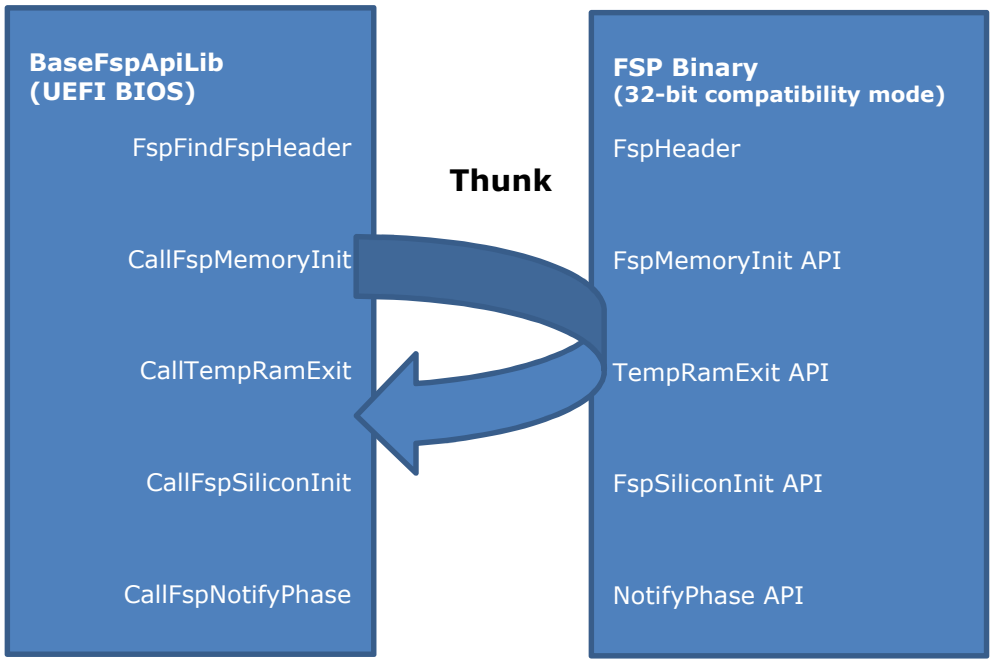
FSP V1.1 provides several APIs for UEFI BIOS to interface with FSP: TempRamInit, FspMemoryInit, TempRamExit, FspSiliconInit and NotifyPhase. UDK2015 FSP Wrapper is responsible for jobs to call above FSP APIs at the proper timing and in proper order.

UDK2015 FSP Wrapper consists of two PEI modules and one DXE driver: FspWrapperSecCore, FspInitPei, and FspNotifyDxe. UEFI BIOS developer have to add these 3 modules into UEFI BIOS codebase.

Above there generic modules also depends on several platform libraries to acquire platform specific information: FspPlatformSecLib, FspPlatformInfoLib, and FspHobProcessLib.

3.4.1 BaseFspApiLib

This library implements thunk interfaces for UEFI BIOS modules to communicate with FSP API. It hides the complexity of locating FSP header and APIs, transitioning processor from long mode to compatibility mode to execute 32-bit FSP code, handoff execution to FSP API, and returning from FSP to UEFI BIOS modules etc.



Note: Please refer to https://svn.code.sf.net/p/edk2/code/branches/UDK2015/IntelFspWrapperPkg/Include/Library/FspApiLib.h for detail declaration of FSP APIs.

3.4.2 FspWrapperSecCore

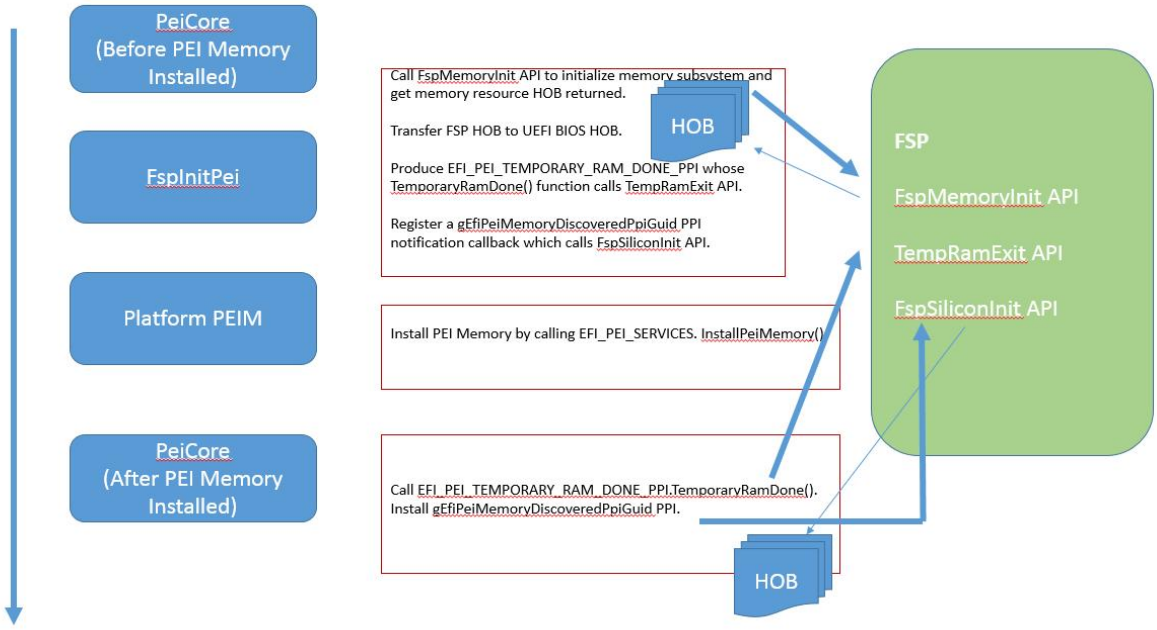
FspWrapperSecCore calls FspTempRamInit API, which will load processor microcode, initializes cache as RAM and so on.

3.4.3 FspInitPei

Entry Point of FspInitPei PEIM mainly does below 3 jobs:

- Calling FspMemoryInit API, parses HOB returned by FspMemoryInit API,
- Registering EFI_PEI_TEMPORARY_DONE_PPI, whose TempRamExit() function calls FSP TempRamExit API,
- Registering gEfiPeiMemoryDiscoveredPpiGuid PPI notification callback, which calls FspSiliconInit API.

Note: The EFI_PEI_TEMPORARY_DONE_PPI. TempRamExit() and gEfiPeiMemoryDiscoveredPpiGuid PPI notification callback will be triggered by PeiCore as soon as PEI Memory gets installed by Platform PEIM by calling EFI_PEI_SERVICES.InstallPeiMemory() .

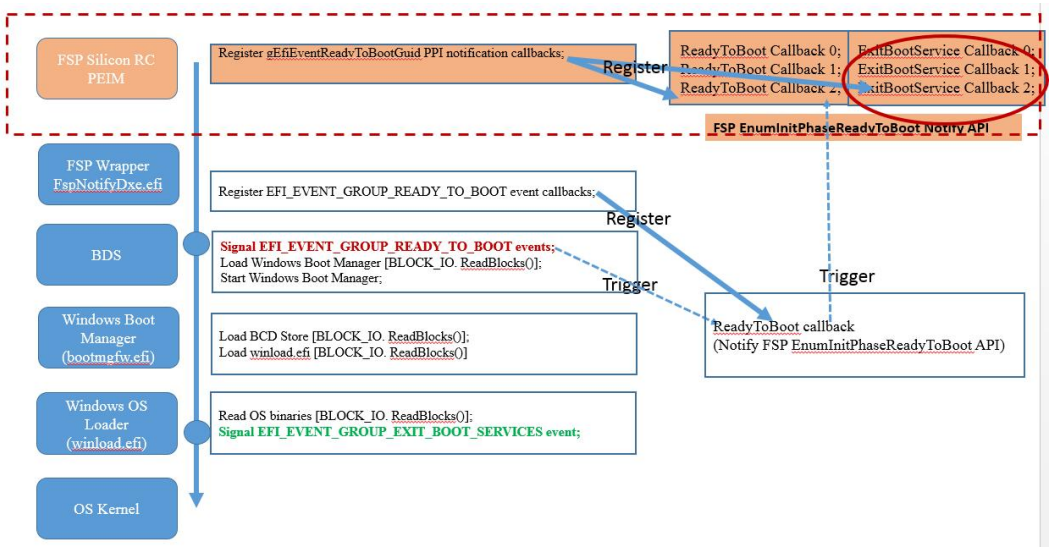


3.4.4 FspNotifyDxe

FspNotifyDxe driver of UDK2015 registers a callback function to *EFI_EVENT_GROUP_READY_TO_BOOT* event, whose job is calling FSP *EnumInitPhaseReadyToBoot NotifyPhase* API.

Note: EFI_EVENT_GROUP_READY_TO_BOOT event is signaled by BDS of UDK2015.

FspNotifyDxe driver also registers a callback function on gEfiPciEnumerationCompleteProtocolGuid protocol notification, whose job is calling FSP *gEfiPciEnumerationCompleteProtocolGuid NotifyPhase* API.



3.4.5 Platform Libraries of FSP Wrapper

Platform libraries, including FspPlatformSecLib, FspPlatformInfoLib, and FspHobProcessLib, will be documented in Chapter 4 - Open Braswell Platform Porting Guide.

4 Open Braswell Platform Porting Guide

4.1 Microcode Integration

Microcode could be integrated into UEFI BIOS as a Firmware Volume (FV). There is no special requirement to base address of Microcode FV. UEFI BIOS could locate this address by PCD gFspWrapperTokenSpaceGuid.PcdCpuMicrocodePatchAddress.

Below sample code of Microcode FV is from Open Braswell platform FDF file.

```
[FD.Cht]
BaseAddress   = $(FLASH_BASE) |
gPlatformModuleTokenSpaceGuid.PcdBiosImageBase
Size          = $(FLASH_SIZE) | gPlatformModuleTokenSpaceGuid.PcdBiosImageSize
…
$(FLASH_REGION_FVMICROCODE_OFFSET)|$(FLASH_REGION_FVMICROCODE_SIZE)
gFspWrapperTokenSpaceGuid.PcdCpuMicrocodePatchAddress|
gFspWrapperTokenSpaceGuid. PcdCpuMicrocodePatchRegionSize
FV = MICROCODE_FV
```

4.2 Firmware Support Package (FSP) Integration

FSP is integrated into UEFI BIOS image as a Firmware Volume (FV). The base address of FSP FV should be either the preferred address documented in FSP Integration Guide, or a rebased address by BCT tool.

Below sample code of FSP FV is from Open Braswell platform FDF file.

```
[FD.Cht]
BaseAddress   = $(FLASH_BASE) |
gPlatformModuleTokenSpaceGuid.PcdBiosImageBase
Size          = $(FLASH_SIZE) | gPlatformModuleTokenSpaceGuid.PcdBiosImageSize
…
$(FLASH_REGION_FSP_OFFSET) | $(FLASH_REGION_FSP_SIZE)
gFspWrapperTokenSpaceGuid.PcdFlashFvFspBase |
gFspWrapperTokenSpaceGuid.PcdFlashFvFspSize
FILE = ChvFspBinPkg/FspBinary/BSWFSP.fd
```

Following two PCDs are consumed by FSP wrapper of Platform Modules to locate FSP.

```
gFspWrapperTokenSpaceGuid.PcdFlashFvFspBase
gFspWrapperTokenSpaceGuid.PcdFlashFvFspSize
```

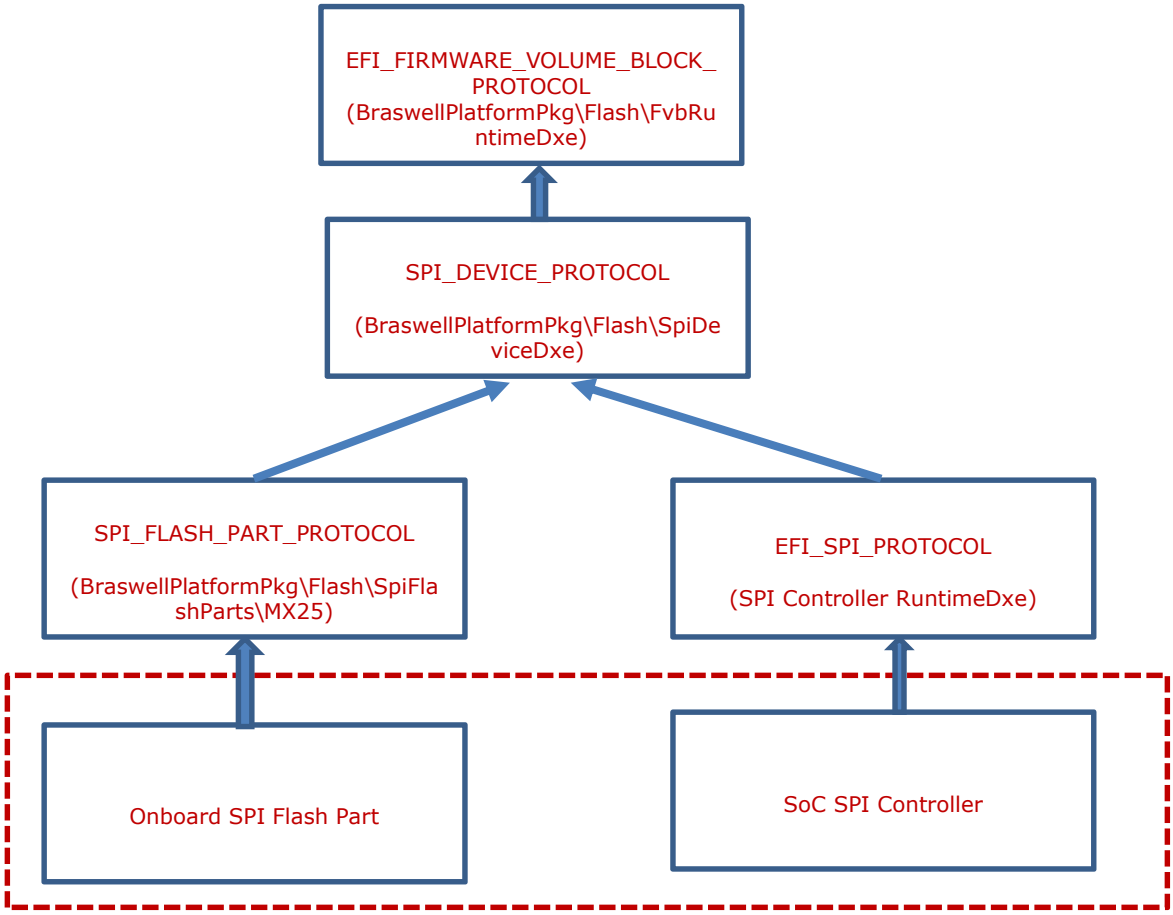
4.3 Video BIOS Table (VBT) Integration

VBT could be integrated into UEFI BIOS image as a Firmware Volume (FV). Several VBT files for different configuration could be included in the single VBT FV.
How to pass into FSP??

4.4 SPI Flash

Braswell SPI Flash sub-system is consists of PCU SPI controller and an onboard SPI flash part. The flash part driver produces SPI_FLASH_PART_PROTOCOL, which provides the operation command supported by this particular flash part. The PCU SPI controller driver produces EFI_SPI_PROTOCOL, which abstracts the SPI communication between controller and device.

While flash part is changed by OEM, only SPI flash part driver need to be rewritten for the new flash part.



4.5 Serial Port Configuration

4.5.1 PlatformHookSerialPortInitialize () API of PlatformHookLib

This API, which is called by SerialPortLib, provides a hook point for initializing platform specific configuration of UART, such as GPIO configuration for UART.

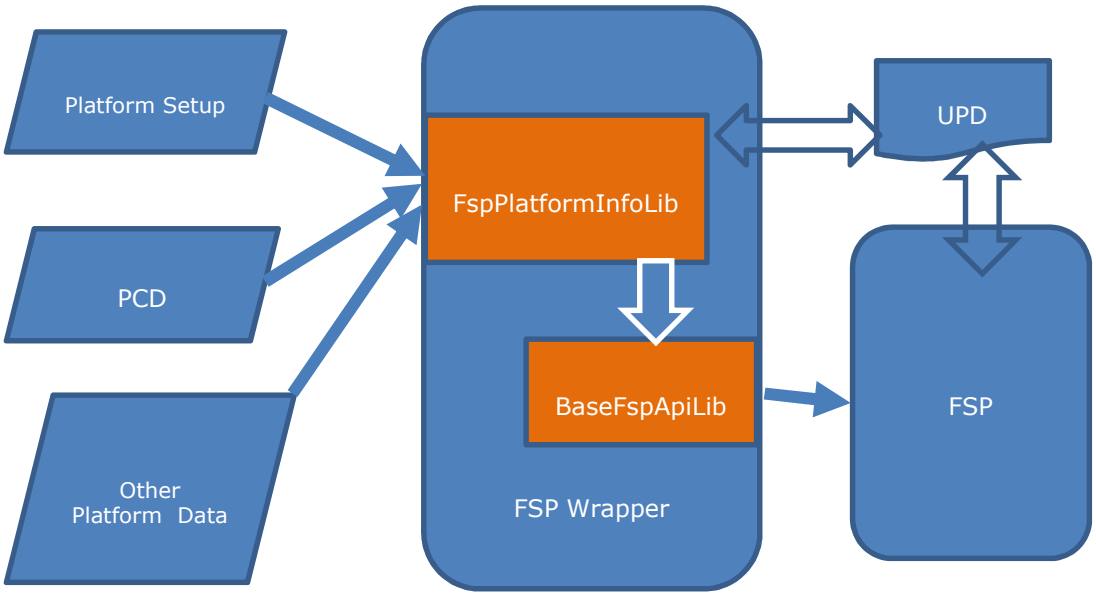
4.5.2 PCDs for SerialPortLib

Following PCD are consumed by SerialPortLib to configure the working mode of serial port.

gEfiMdePkgTokenSpaceGuid.PcdUartDefaultBaudRate
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialBaudRate
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseHardwareFlowControl
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseMmio
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialRegisterBase
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialLineControl
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialFifoControl
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseHardwareFlowControl
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialDetectCable
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialRegisterStride
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialClockRate
gEfiMdeModulePkgTokenSpaceGuid.PcdSerialPciDeviceInfo

4.6 Platform Configuration for FSP APIs

Platform library FspPlatformInfoLib for FSP wrapper provides hooks for FSP wrapper to convert platform Setup, PCD and other platform specific data to be input parameter of FSP APIs or to update FSP UPD region dynamically. Please refer to appendix A for the sample definition of Braswell FSP UPD.



4.6.1 Platform Configuration for TempRamInit API

FSP wrapper FspWrapperSecCore requires following platform PCDs to feed FSP TempRamInit API. Please refer to section 4.1 for how *PcdCpuMicrocodePatchAddress* and *PcdCpuMicrocodePatchRegionSize* are assigned value by platform FDF file. It is up to BIOS developers to determine the value of *PcdPeiTemporaryRamStackSize*.

```
gFspWrapperTokenSpaceGuid.PcdPeiTemporaryRamStackSize
gFspWrapperTokenSpaceGuid.PcdFlashCodeCacheAddress
gFspWrapperTokenSpaceGuid.PcdFlashCodeCacheSize
gFspWrapperTokenSpaceGuid.PcdCpuMicrocodePatchAddress
gFspWrapperTokenSpaceGuid.PcdCpuMicrocodePatchRegionSize
gFspWrapperTokenSpaceGuid.PcdFlashMicroCodeOffset
```

4.6.2 Platform Configuration for FspMemoryInit API

UpdateFspUpdConfigs () API of FspPlatformInfoLib provides a platform hook point for FSP Wrapper FspInitPei to update MEMORY_INIT_UPD for FspMemoryInit API.

Platform Setup and PCD information could be used to update MEMORY_INIT_UPD by UpdateFspUpdConfigs (). (List several common item here, such as SPD.)

Note: Please refer to Appendix A for sample data structure of MEMORY_INIT_UPD.

We provide following sample code for UpdateFspUpdConfigs () API.

```
VOID * EFIAPI UpdateFspUpdConfigs (IN OUT VOID       *FspUpdRgnPtr) {
    FSP_INFO_HEADER          *pFspHeader = NULL;
    VPD_DATA_REGION          *FspVpdRgn = NULL;
    UINT32                    UpdRegionSize = sizeof(UPD_DATA_REGION);
    MEMORY_INIT_UPD          *MemoryInitUpd = NULL;
    UPD_DATA_REGION          *pFspUpdRgnPtrTmp = NULL;

    if (NULL == FspUpdRgnPtr)  return NULL;
    pFspUpdRgnPtrTmp = (UPD_DATA_REGION *)FspUpdRgnPtr;
    if (PcdGet32 (PcdFlashFvSecondFspBase) == 0) {
      pFspHeader = FspFindFspHeader (PcdGet32 (PcdFlashFvFspBase));
    } else {
      pFspHeader = FspFindFspHeader (PcdGet32 (PcdFlashFvSecondFspBase));
    }
    if (NULL == pFspHeader) { return NULL; }
    FspVpdRgn = (VPD_DATA_REGION *)(UINTN)(pFspHeader->ImageBase
                    + pFspHeader->CfgRegionOffset);
  CopyMem (
     FspUpdRgnPtr,
    (void *)(pFspHeader->ImageBase + FspVpdRgn->PcdUpdRegionOffset),
    UpdRegionSize);
  MemoryInitUpd = (MEMORY_INIT_UPD *)((UINT8 *)FspUpdRgnPtr +
                      pFspUpdRgnPtrTmp->MemoryInitUpdOffset);
  UpdateMemoryInitUpd((MEMORY_INIT_UPD *)MemoryInitUpd);
```

```
  return (VOID *)MemoryInitUpd;
}

STATIC  EFI_STATUS  UpdateMemoryInitUpd (
  MEMORY_INIT_UPD       *FspUpdRgn
) {
  MEMORY_INIT_UPD *MemoryInitUpd;
  MemoryInitUpd = (MEMORY_INIT_UPD *)FspUpdRgn;
  MemoryInitUpd->PcdMemChannel0Config = PcdGet8(PcdOemMemeoryDimmType);
  MemoryInitUpd->PcdMemChannel1Config = PcdGet8(PcdOemMemeoryDimmType);
  MemoryInitUpd->PcdMrcInitSpdAddr1 = PcdGet8(PcdMrcInitSpdAddr1);
  return EFI_SUCCESS;
}
```

4.6.3 Platform Configuration for FspSiliconInit API

GetFspSiliconInitParam() API of *FspPlatformInfoLib* provides a platform hook point for FSP Wrapper FspInitPei to update *SILICON_INIT_UPD* for FspSiliconInit API.

Platform Setup and PCD information could be used to update *SILICON_INIT_UPD* by *GetFspSiliconInitParam ().* (List several common item here, such as GPIO ….)

Please refer to Appendix A for data structure of *SILICON_INIT_UPD*.

We provide following sample code for *GetFspSiliconInitParam()* API.

```
VOID *
EFIAPI
GetFspSiliconInitParam ( VOID )
{
  FSP_INFO_HEADER          *FspHeader;
  VPD_DATA_REGION          *FspVpdRgn;
  UPD_DATA_REGION          *FspUpdRgnPtr;
  SILICON_INIT_UPD         *FspSiliconInitUpd;
  EFI_PEI_SERVICES         **PeiServices;

  if (PcdGet32 (PcdFlashFvSecondFspBase) == 0) {
    FspHeader = FspFindFspHeader (PcdGet32 (PcdFlashFvFspBase));
  } else {
    FspHeader = FspFindFspHeader (PcdGet32 (PcdFlashFvSecondFspBase));
  }
  if (FspHeader == NULL) { return NULL;}

  PeiServices = (EFI_PEI_SERVICES **)GetPeiServicesTablePointer ();
  if (PeiServices == NULL) { return NULL;}

  //Get VPD region start
  FspVpdRgn = (VPD_DATA_REGION *)(UINTN)(FspHeader->ImageBase
                + FspHeader->CfgRegionOffset);
  ASSERT (PcdGet32 (PcdMaxUpdRegionSize) >= sizeof(UPD_DATA_REGION));
  FspUpdRgnPtr = (UPD_DATA_REGION *)AllocatePool (
                   PcdGet32 (PcdMaxUpdRegionSize));
  if (NULL == FspUpdRgnPtr) { return NULL;}
```

```
  CopyMem ((UINT8 *)FspUpdRgnPtr, (UINT8 *)(FspHeader->ImageBase
              + FspVpdRgn->PcdUpdRegionOffset), sizeof(UPD_DATA_REGION));
  FspSiliconInitUpd = (SILICON_INIT_UPD *)((UINT8 *)FspUpdRgnPtr
                      + FspUpdRgnPtr->SiliconInitUpdOffset);

  UpdateSilicionInitUpd (PeiServices, FspSiliconInitUpd);

  return FspSiliconInitUpd;
}

STATIC
EFI_STATUS
UpdateSilicionInitUpd (
 IN EFI_PEI_SERVICES  **PeiServices,
 IN SILICON_INIT_UPD  *SiliconInitUpd
)
{
  SiliconInitUpd->PMIC_I2CBus = 0;
  SiliconInitUpd->PcdEnableHsuart0 = SystemConfiguration.LpssHsuart0Enabled;
  SiliconInitUpd->PcdEnableHsuart1 = SystemConfiguration.LpssHsuart1Enabled;
}
```

4.6.4 Platform HOB

FspHobProcessForMemoryResource() API of platform FspHobProcessLib provides a hook point for FSP Wrapper FspInitPei to retrieve HOBs returned by FspMemoryInit API.

EFI_HOB_TYPE_RESOURCE_DESCRIPTOR
EFI_RESOURCE_SYSTEM_MEMORY
EFI_RESOURCE_MEMORY_RESERVED

FspHobProcessForOtherData() API of platform FspHobProcessLib provides a hook point for FSP Wrapper FspInitPei to retrieve HOBs returned by FspSiliconInit API.

EFI_HOB_TYPE_GUID_EXTENSION
gFspNonVolatileStorageHobGuid
gFspBootLoaderTemporaryMemoryGuid
gEfiPlatformCpuInfoGuid
gEfiVariableGuid
gEfiSmmPeiSmramMemoryReserveGuid
gEfiAcpiVariableGuid
gFspSmbiosMemoryInfoHobGuid
gEfiGraphicsInfoHobGuid
gEfiPlatformInfoGuid

4.7 SMI Handler Register

The *PchSmiDispatcher* module produces SMI handler registration services for Braswell South Cluster generated SMIs, which are defined by PI1.4 specification Volume4

System Management Mode Core Interface. Platform drivers could leverage these SMM dispatch protocol to register platform specific SMI handlers.

EFI_SMM_USB_DISPATCH2_PROTOCOL

EFI_SMM_SX_DISPATCH2_PROTOCOL

EFI_SMM_SW_DISPATCH2_PROTOCOL

EFI_SMM_GPI_DISPATCH2_PROTOCOL

EFI_SMM_ICHN_DISPATCH_PROTOCOL

EFI_SMM_POWER_BUTTON_DISPATCH2_PROTOCOL

EFI_SMM_PERIODIC_TIMER_DISPATCH2_PROTOCOL

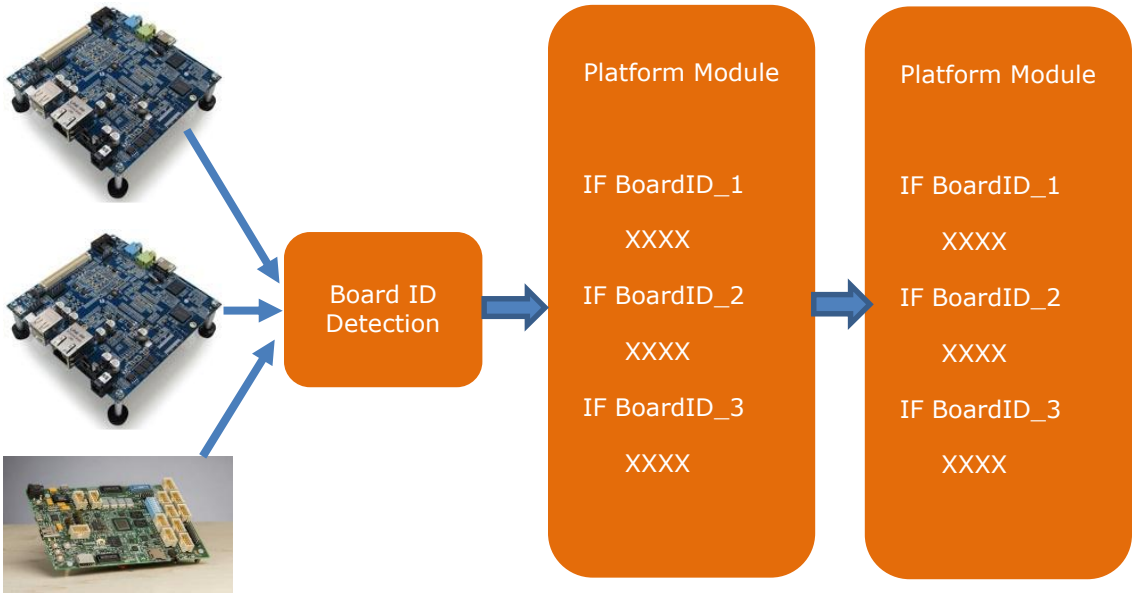
4.8 ACPI Tables (TODO)

This module contains the ASL files for the Braswell SOC which includes the Processor, North Cluster and South Cluster related devices.

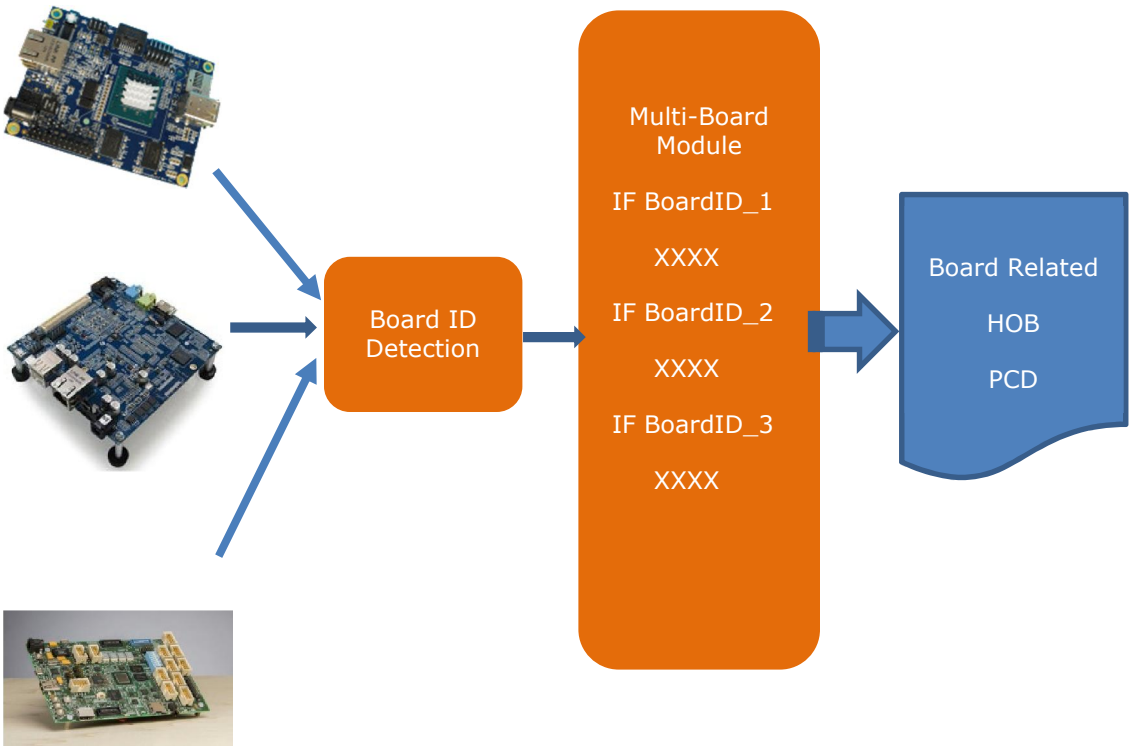
4.9 Multi-Board Support

Braswell Platform provides the flexibility for OEM to design their own special board. In order to make a single BIOS codebase support different board configuration, platform module has to detect the Board ID, which may be represented simply by an array of GPIO pin combination, and then do special configuration for that particular board.

There are two ways to make codebase support multi-boards. The first one is like below illustration showed. In order to support multi-board case, a platform modules use "if" statements to handle different boards. The disadvantage of this solution is: All those board related platform modules need to be updated if a new board is added into codebase.



The second solution of supporting multi-boards is creating a new multi-board module, which collects all platform data and save them in HOBs or PCDs for subsequent platform modules to consume. The advantage of this solution is: If OEM wants to add a new board into codebase, only the multi-board module needs to be updated and all other platform modules keep unchanged. Open Braswell codebase takes this solution.



Please refer to following multi-board modules in Open Braswell codebase:

```
BraswellPlatformPkg\
  Board\
     BraswellCR\
        BoardInit\
           BoardInit.inf
     CherryHill\
        BoardInit\
           BoardInit.inf
```

For ACPI ASL code, we create an ACPI SSDT table for each board and move each board specific ASL code into its corresponding SSDT. And then the Board ID is used to decide which SSDT will be installed into ACPI at runtime. Please refuter to following modules for ACPI SSDT:

```
BraswellPlatformPkg\
  Board\
     BraswellCR\
        Acpi\
           Acpi.inf
     CherryHill\
        Acpi\
           Acpi.inf
```

5 Boot Flow of Open Braswell UEFI BIOS

The normal code flow of Open Braswell EDKII codebase passes through a succession of phases in the following order:

SEC

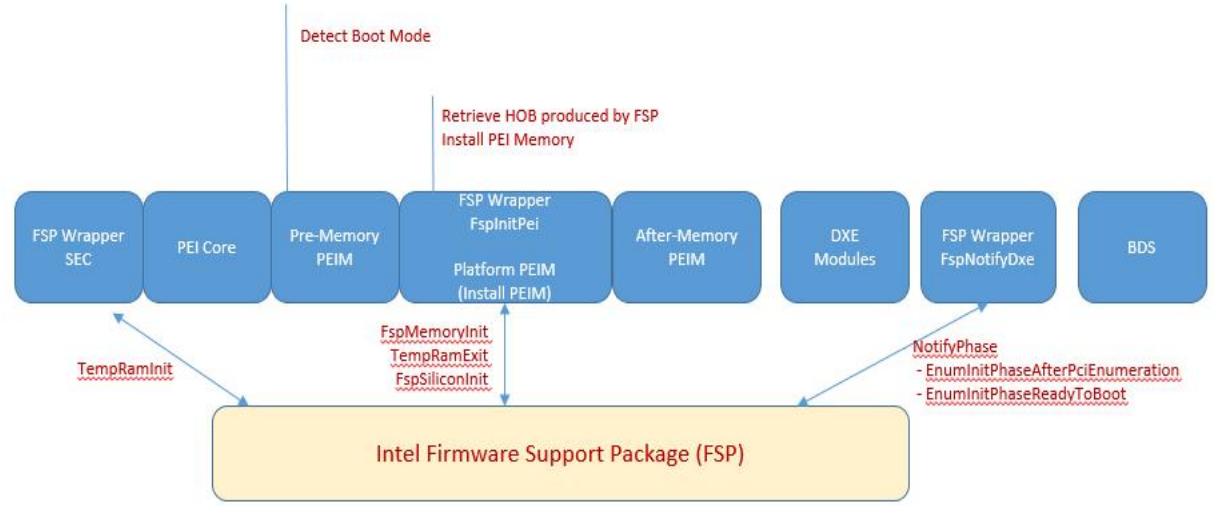
PEI

DXE

BDS

Runtime

In order to initialize Braswell SoC basic silicon functions, Open Braswell EDKII FSP Wrapper is responsible for communicating with Braswell FSP APIs in SEC, PEI, and BDS phase. The following diagram illustrates the high level boot flow.



6 Appendix B - VPD and UPD definition of Braswell FSP

FspVpdUpd.h file of Braswell FSP release package.

```
#pragma pack(1)
#define MAX_CHANNELS_NUM 2
#define MAX_DIMMS_NUM    2

typedef struct {
  UINT32  VendorDeviceId;
  UINT16  SubSystemId;
  UINT8   RevisionId;                 /// 0xFF applies to all steppings
  UINT8   FrontPanelSupport;
  UINT16  NumberOfRearJacks;
  UINT16  NumberOfFrontJacks;
} BL_PCH_AZALIA_VERB_TABLE_HEADER;

typedef struct {
  BL_PCH_AZALIA_VERB_TABLE_HEADER  VerbTableHeader;
  UINT32                    *VerbTableData;
} BL_PCH_AZALIA_VERB_TABLE;

typedef struct {
  UINT8             Pme      : 1;     /// 0: Disable; 1: Enable
  UINT8             DS       : 1;     /// 0: Docking is not supported; 1:Docking is
supported
  UINT8             DA       : 1;     /// 0: Docking is not attached; 1:Docking is
attached
  UINT8             HdmiCodec : 1;     /// 0: Disable; 1: Enable
  UINT8             AzaliaVCi : 1;     /// 0: Disable; 1: Enable
  UINT8             Rsvdbits  : 3;
  UINT8             AzaliaVerbTableNum; /// Number of verb tables provided by
platform
  BL_PCH_AZALIA_VERB_TABLE  *AzaliaVerbTable;   /// Pointer to the actual verb
table(s)
  UINT16             ResetWaitTimer;     /// The delay timer after Azalia reset, the
value is number of microseconds
} BL_PCH_AZALIA_CONFIG;

typedef struct {
  UINT32          Confg;
  UINT32          ConfgChanges;
  UINT32          Misc;
  UINT32          MmioAddr;
  CHAR16           *Name;
} BL_GPIO_FAMILY_INIT;

typedef struct {
  UINT32          Confg0;
```

```
  UINT32          Confg0Changes;
  UINT32          Confg1;
  UINT32          Confg1Changes;
  UINT32          Community;
  UINT32          MmioAddr;
  CHAR16          *Name;
  UINT32          Misc;
} BL_GPIO_PAD_INIT;

typedef struct {
  UINT8        DimmId;
  UINT32       SizeInMb;
  UINT16       MfgId;
  UINT8        ModulePartNum[20];
} DIMM_INFO;

typedef struct {
  UINT8        ChannelId;
  UINT8        DimmCount;
  DIMM_INFO    DimmInfo[MAX_DIMMS_NUM];
} CHANNEL_INFO;

typedef struct {
  UINT8        Revision;
  UINT8        DataWidth;
  UINT8        MemoryType;
  UINT16       MemoryFrequencyInMHz;
  UINT8        ErrorCorrectionType;
  UINT8        ChannelCount;
  CHANNEL_INFO  ChannelInfo[MAX_CHANNELS_NUM];
} FSP_SMBIOS_MEMORY_INFO;


typedef struct {
  UINT64                Signature;
  UINT8                 Revision;
  UINT8                 UnusedUpdSpace2[7];
  UINT16                PcdMrcInitTsegSize;
  UINT16                PcdMrcInitMmioSize;
  UINT8                 PcdMrcInitSpdAddr1;
  UINT8                 PcdMrcInitSpdAddr2;
  UINT8                 PcdMemChannel0Config;
  UINT8                 PcdMemChannel1Config;
  UINT32                PcdMemorySpdPtr;
  UINT8                 PcdIgdDvmt50PreAlloc;
  UINT8                 PcdApertureSize;
  UINT8                 PcdGttSize;
  UINT8                 PcdLegacySegDecode;
  UINT8                 PcdDvfsEnable;
  UINT8                 PcdMemoryTypeEnable;
  UINT8                 PcdCaMirrorEn;
  UINT8                 ReservedMemoryInitUpd[189];
} MEMORY_INIT_UPD;
```

```
typedef struct {
  UINT64                  Signature;
  UINT8                   Revision;
  UINT8                   UnusedUpdSpace3[7];
  UINT8                   PcdSdcardMode;
  UINT8                   PcdEnableHsuart0;
  UINT8                   PcdEnableHsuart1;
  UINT8                   PcdEnableAzalia;
  BL_PCH_AZALIA_CONFIG*       AzaliaConfigPtr;
  UINT8                   PcdEnableSata;
  UINT8                   PcdEnableXhci;
  UINT8                   PcdEnableLpe;
  UINT8                   PcdEnableDma0;
  UINT8                   PcdEnableDma1;
  UINT8                   PcdEnableI2C0;
  UINT8                   PcdEnableI2C1;
  UINT8                   PcdEnableI2C2;
  UINT8                   PcdEnableI2C3;
  UINT8                   PcdEnableI2C4;
  UINT8                   PcdEnableI2C5;
  UINT8                   PcdEnableI2C6;
  UINT32                  GraphicsConfigPtr;
  BL_GPIO_FAMILY_INIT*        GpioFamilyInitTablePtr;
  BL_GPIO_PAD_INIT*           GpioPadInitTablePtr;
  UINT8                   PunitPwrConfigDisable;
  UINT8                   ChvSvidConfig;
  UINT8                   DptfDisable;
  UINT8                   PcdEmmcMode;
  UINT8                   PcdUsb3ClkSsc;
  UINT8                   PcdDispClkSsc;
  UINT8                   PcdSataClkSsc;
  UINT8                   Usb2Port0PerPortPeTxiSet;
  UINT8                   Usb2Port0PerPortTxiSet;
  UINT8                   Usb2Port0IUsbTxEmphasisEn;
  UINT8                   Usb2Port0PerPortTxPeHalf;
  UINT8                   Usb2Port1PerPortPeTxiSet;
  UINT8                   Usb2Port1PerPortTxiSet;
  UINT8                   Usb2Port1IUsbTxEmphasisEn;
  UINT8                   Usb2Port1PerPortTxPeHalf;
  UINT8                   Usb2Port2PerPortPeTxiSet;
  UINT8                   Usb2Port2PerPortTxiSet;
  UINT8                   Usb2Port2IUsbTxEmphasisEn;
  UINT8                   Usb2Port2PerPortTxPeHalf;
  UINT8                   Usb2Port3PerPortPeTxiSet;
  UINT8                   Usb2Port3PerPortTxiSet;
  UINT8                   Usb2Port3IUsbTxEmphasisEn;
  UINT8                   Usb2Port3PerPortTxPeHalf;
  UINT8                   Usb2Port4PerPortPeTxiSet;
  UINT8                   Usb2Port4PerPortTxiSet;
  UINT8                   Usb2Port4IUsbTxEmphasisEn;
  UINT8                   Usb2Port4PerPortTxPeHalf;
  UINT8                   Usb3Lane0Ow2tapgen2deemph3p5;
  UINT8                   Usb3Lane1Ow2tapgen2deemph3p5;
  UINT8                   Usb3Lane2Ow2tapgen2deemph3p5;
  UINT8                   Usb3Lane3Ow2tapgen2deemph3p5;
```

```
  UINT8                 PcdSataInterfaceSpeed;
  UINT8                 PcdPchUsbSsicPort;
  UINT8                 PcdPchUsbHsicPort;
  UINT8                 PcdPcieRootPortSpeed;
  UINT8                 PcdPchSsicEnable;
  UINT32                PcdLogoPtr;
  UINT32                PcdLogoSize;
  UINT8                 PcdRtcLock;
  UINT8                 PMIC_I2CBus;
  UINT8                 ISPEnable;
  UINT8                 ISPPciDevConfig;
  UINT8                 PcdTurboMode;
  UINT8                 PcdPnpSettings;
  UINT8                 PcdSdDetectChk;
  UINT8                 ReservedSiliconInitUpd[411];
} SILICON_INIT_UPD;

#define FSP_UPD_SIGNATURE           0x2444505557534224      /* '$BSWUPD$' */
#define FSP_MEMORY_INIT_UPD_SIGNATURE  0x244450554D454D24      /*
'$MEMUPD$' */
#define FSP_SILICON_INIT_UPD_SIGNATURE   0x244450555F495324      /*
'$SI_UPD$' */

typedef struct _UPD_DATA_REGION {
  UINT64                Signature;
  UINT8                 Revision;
  UINT8                 UnusedUpdSpace0[7];
  UINT32                MemoryInitUpdOffset;
  UINT32                SiliconInitUpdOffset;
  UINT64                UnusedUpdSpace1;
  MEMORY_INIT_UPD           MemoryInitUpd;
  SILICON_INIT_UPD          SiliconInitUpd;
  UINT16                PcdRegionTerminator;
} UPD_DATA_REGION;

#define FSP_IMAGE_ID    0x2450534657534224      /* '$BSWFSP$' */
#define FSP_IMAGE_REV   0x01010200

typedef struct _VPD_DATA_REGION {
  UINT64                PcdVpdRegionSign;
  UINT32                PcdImageRevision;
  UINT32                PcdUpdRegionOffset;
} VPD_DATA_REGION;
#pragma pack()
```