White Paper

A Tour Beyond BIOS Open Source IA Firmware Platform Design Guide in EFI Developer Kit II (version 2) -- OpenKabylake as example

Jiewen Yao
Intel Corporation

Vincent J. Zimmer
Intel Corporation

Michael A Kubacki
Intel Corporation

Amy Chan
Intel Corporation

Rangasai V Chaganty
Intel Corporation

Chasel Chiu
Intel Corporation

May 2017

Executive Summary

This paper introduces a design guide for an EDKII open source IA firmware solution. In order to make an open IA firmware solution simple, we demonstrate a firmware design approach with minimal features. The only criteria are 1) It can boot to the OS, and 2) It is secure. We can remove many unnecessary silicon or platform features like Capsule update, Recovery, S3 resume, SMBIOS, Embedded Controller (EC), Super IO (SIO), I2C, and only enable ACPI & SMM to support booting.

Prerequisite

This paper assumes that the audience has EDKII/UEFI firmware development experience [UEFI][UEFI PI Specification] and Intel® FSP knowledge [FSP]. He or she should be familiar with the UEFI/PI firmware infrastructure (e.g., PEI/DXE) [UEFI Book] and understand the Intel FSP flow [FSP EAS] [FSP Consumer].

Table of Contents

Overview

Introduction to open source Intel Architecture (IA) firmware

In order to make an IA platform boot, the IA firmware is needed to initialize the silicon and report necessary information to an operating system. An open source IA firmware is a firmware solution having public silicon code and public platform code based upon public specifications, such as [IA32 Manual] [Intel Graphic OpRegion] [Intel TXT] [Intel SGX] [Intel TraceHub] [Intel VT-d] [Baytrail data sheet] [Brasswell data sheet] [Quark data sheet] [Skylake SA data sheet] [Skylake PCH data sheet] [Kabylake SA data sheet] [Kabylake PCH data sheet]. The only binaries should be CPU microcode and the Intel Firmware Support Package (FSP) binary, each which contain IP sensitive content. The benefit of an open source IA firmware solution is that everyone in the world can take the open source IA firmware as an example starting point, build a new platform, and subsequently create a new firmware solution.
Currently the open source IA firmware infrastructure includes [COREBOOT] and [EDK2]

Introduction to EDKII

EDKII is an open source implementation of UEFI PI-based firmware which can boot multiple UEFI-aware operating systems. The EDKII open source project includes several open source IA firmware such as MinnowBoard MAX, Quark, Broxton, Kabylake. There will be more in the future.

Summary
This section provided an overview of an open platform firmware solution and EDKII.

Open Source IA Firmware Design

Problem statement

Before an open source IA firmware appeared, there were many closed source IA firmware solutions in the world. We did research on the UEFI firmware examples of Intel ATOM based small core, Intel Core-i7 based big core client, and Intel XEON based big core servers. We came across some common issues, including:

1) Developers need a way to turn on and off of a feature.

   For example, the Trusted Platform Module (TPM) or UEFI Secure Boot may need to be supported. In principle it is valuable to provide this class of capability, but the problem is too many configurations are often provided. In fact, we observed that one BIOS provides more than 100 configurations exposed for the developers to control. Regrettably, some configurations of those various controls do not even work.

   This often leads to the request of having a minimal BIOS to boot, and to do so, how to select among these 100 configurations?

2) Developers need a way to get the platform configuration data.

   For example, one configuration choice can include “is VT enabled by the end user?” Another control can include “is the TSEG SMRAM size 1M, 8M or 16M,” or “is there an Embedded Controller (EC) or DOCK attached on the board?” The EDKII BIOS provides many choices on the source of the configuration data. For example, the UEFI specification defines UEFI Variables; the UEFI PI specification defines PCD; the Intel FSP defines VPD and UPD; silicon reference code defines the policy HOB, policy PPI, and policy protocol; silicon specific code has a signed static configuration data blob; and even legacy CMOS region exists.

   People may ask: which interface should I use in my platform code?

3) Developers need to do porting work from an existing board to a new board.

   There might be GPIO routing differences, or alternate component choices, such as SIO differences. However, some older platform code may use a “switch-case” mechanisms to check the board type, with such “switch-case” usages is scattered across many platform drivers. These platform elements include AcpiPlatform, SmmPlatform, PlatformInit, EC, ASL code, VFR pages, etc. In order to add a new board on update a existing platform, a developer has to find out all the places to make the change.

   People may think: How can I know how many modules I need to port, and when have I finished updating all required modules?

4) Developers might need to work on a different board.

For example, there might be an ATOM based on a server, a Core-i7 based server, or a XEON based server. However, the BIOS from different segments are different. We once compared an ATOM based firwmare with a Core-i7 based firmware. There are ~20 directories under Platform. Only 2 are same, which are "Include", and "Library". People might require significant time to ramp up again to get familiar with the new platform structure.

Why can't the platform tree structures bear more similarity?

Goal

Based on the above observations, people may feel that the existing IA firmware is complex and hard to port or enable for a new platform. The purpose of this whitepaper is to provide some guidance on how to design an IA firmware solution to meet the goals below:

- **Simple**. Code structure should be obvious so that the firmware developer can easily turn on or turn off a significant feature.

- **Portable**. Firmware developer can easily port and enable a new board.

- **Consistent**. Firmware code structure should be similar, no matter if it is an ATOM based embedded platform, a Core-i7 based mobile device, or a XEON based server.

Focus

In order to provide suggestions on the problem statements above, we would like to focus on the following four areas:

- **Feature**. How does a BIOS provide the feature selection option to a developer?
- **Configuration**. From which interface can a platform module get the configuration data?
- **Porting**. Where are the modules to be ported for a new board?
- **Tree Structure**. What does an EDKII platform package look like?

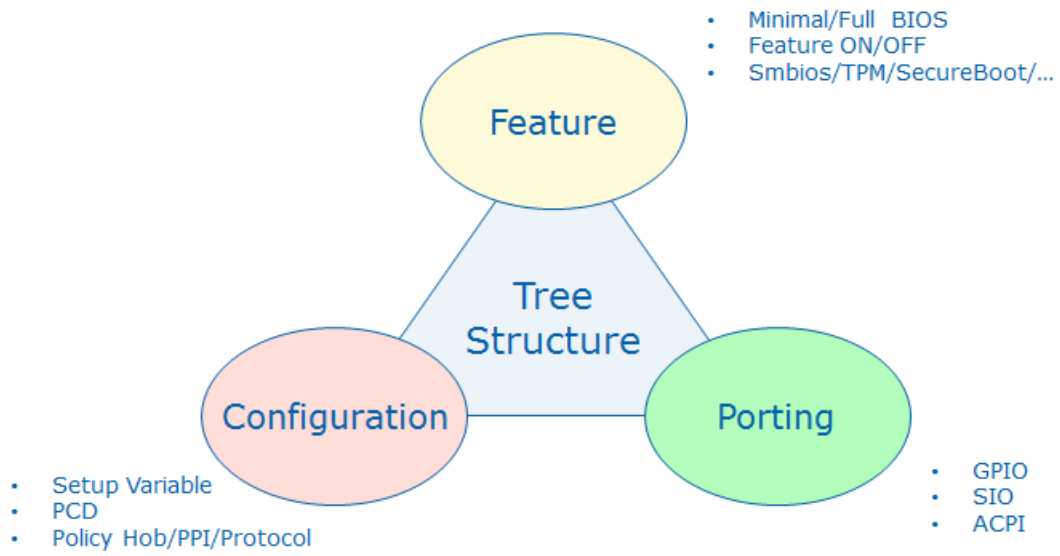

Figure 1 Focus Area

In the next several chapters, we will use some real examples to demonstrate these design ideas. We will use OpenKabylake as an example.

The code is @ https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform and https://github.com/tianocore/edk2-non-osi/tree/devel-MinPlatform

The associated Intel FSP is located at https://github.com/IntelFsp/FSP/tree/Kabylake.

Summary
This section introduces the open source IA firmware design goals.

Tree Structure – Platform Code Layout

This section discusses the tree structure.

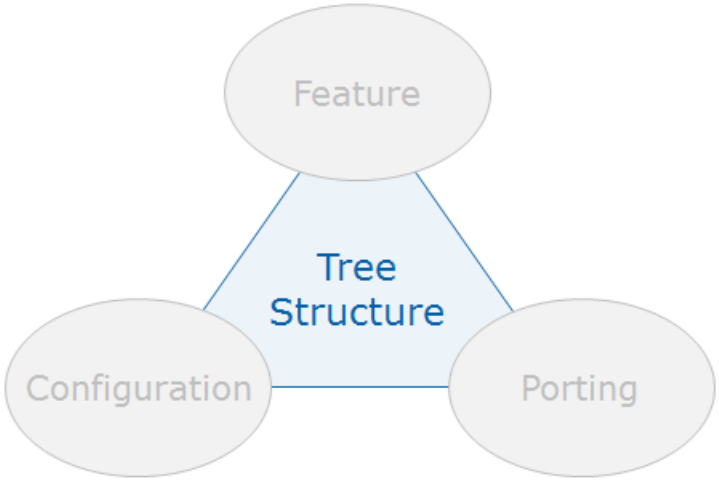

Figure 2 Focus Area – Tree Structure

Open Board Tree Structure

The whole package structure is shown below:

```
====== Tree Structure ======
edk2-platform/ (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform)
      Platform/
            Intel/
                  KabylakeOpenBoardPkg
                  MinPlatformPkg
      Silicon/
            Intel/
                  KabylakeSiliconPkg
edk2-non-osi/ (https://github.com/tianocore/edk2-non-osi/tree/devel-MinPlatform)
      Silicon/
            Intel/
                  KabylakeSiliconBinPkg
FSP/
      KabylakeFspBinPkg
====== Tree Structure ======
```

Below includes the role and responsibility of each folder.

- **edk2-platform**: It is the EDKII repo to include any open source platform code, which has an edkII compatible license.
  - **Platform** folder: It contains the platform specific modules.
  - **Silicon** folder: It contains the silicon specific modules.
  - **MinPlatformPkg**: It is a generic platform instance to control the boot flow.

    - **<Generation>OpenBoardPkg**: It is the silicon generation specific board package. All of the boards based upon this silicon generation can be located here.
    - **<Generation>SiliconPkg**: It is the silicon generation specific silicon package.
- **edk2-non-osi**: It is the EDKII repo to include any open platform modules which is in a binary format, such as FSP binary, or CPU microcode.
    - **<Generation>SiliconBinPkg**: It is the silicon generation specific binary package. For example, CPU Microcode can be found here.

Ideally, when we port a new board, only the code under **<Generation>OpenBoardPkg** needs to be updated.

Board Package Structure

The MinPlatformPkg has the below-listed structure:

```
====== Package Structure ======
MinPlatformPkg /
      <BasicCommonFeature>/
      Include
      Library
      Features
            <AdvancedCommonFeatures>
      PlatformInit
====== Package Structure ======
```

Below describes the purpose of each folder.

- **<BasicCommonFeature>**: The basic features to support OS boot, such as ACPI, flash, and FspWrapper. It also includes the basic security features such as HSTI.
- **Include**: The include file as the package interface. All interfaces defined in MinPlatformPkg.dec are put to here.
- **Library**: It only contains feature independent library, such as PeiLib. If a library is related to a feature, this library is put to <Feature>/Library folder, instead of root Library folder.
- **Features/< AdvancedCommonFeatures>**: The advanced features.
- **PlatformInit**: The common platform initialization module. There is PreMemPEI, PostMemPEI, DXE and SMM version. These modules control boot flow and provide some hook point to let board code do initialization.

The <Generation>OpenBoardPkg has similar structure to MinPlatformPkg.

```
====== Package Structure ======
<Generation>OpenBoardPkg/
      <BasicCommonBoardFeature>/
      Include
      Library
      Features
            <AdvancedCommonBoardFeatures>
      <Board>
            Include/
            Library/
            <BoardSpecificFeature>
```

```
====== Package Structure ======
```

- The **<BasicCommonBoardFeature>** and **<AdvancedCommonBoardFeatures>** designate a board generation specific feature. They need to be updated when we enable a board generation.
- The **<Board>** folder contains all the board specific settings. If we need to port a new board in this generation, we can copy the <Board> folder and update the settings there.

Guideline: One feature, one directory

We observed that some platforms use a flat directory layout and put most modules in the Platform directory. This introduces issues when the developer wants to find a driver. Below is an example for the open source MinnowMAX platform directory.

https://github.com/tianocore/edk2/tree/master/Vlv2TbltDevicePkg (GIT-HASH: 6b49f0e0d36e926042d91d2c78066b3d529c739f)

```
====== Vlv2TbltDevicePkg directory ======
Vlv2TbltDevicePkg
  AcpiPlatform
  Application
  BootScriptSaveDxe
  FspAzaliaConfigData
  FspSupport
  FvbRuntimeDxe
  FvInfoPei
  Include
  IntelGopDepex
  Library
  Logo
  Metronome
  MonoStatusCode
  Override
  PciPlatform
  PlatformCpuInfoDxe
  PlatformDxe
  PlatformGopPolicy
  PlatformInfoDxe
  PlatformInitPei
  PlatformPei
  PlatformSetupDxe
  PlatformSmm
  PpmPolicy
  SaveMemoryConfig
  SmBiosMiscDxe
  SmmSwDispatch2OnSmmSwDispatchThunk
  SmramSaveInfoHandlerSmm
  Stitch
  UiApp
  VlvPlatformInitDxe
  Wpce791
====== Vlv2TbltDevicePkg directory ======
```

We recommend using a hierarchical layout. Specifically, only put the basic features into the root directory and put the advanced features into a “Features” directory.

Below is MinPlatformPkg (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg). The ACPI related features are in the Acpi directory. Flash driver – SpiFvbService is under the Flash directory. FspWrapper related modules are in FspWrapper. Test folder contains some platform test point related modules. Finally, PlatformInit directory includes PlatformInitPei, PlatformInitDxe and PlatformInitSmm.

```
====== MinPlatformPkg directory ======
MinPlatformPkg
  Acpi
      AcpiSmm
      AcpiTables
  Flash
      SpiFvbService
  FspWrapper
      Library
      SaveMemoryConfig
  Hsti
  Include
  Library
      PeiLib
  PlatformInit
      PlatformInitDxe
      PlatformInitPei
      PlatformInitSmm
  Test
      Library
          TestPointCheckLib
          TestPointLib
      TestPointDumpApp
  Tools
====== MinPlatformPkg directory ======
```

Below is the KabylakeOpenBoardPkg (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg). The common board related ACPI is in Acpi directory. The common board related FSP policy update is in FspWrapper. The library folder includes the board specific GpioExpanderLib and I2cAccessLib. They might be used for other Kabylake generation board.
The KabylakeRvp3 folder contains all RVP3 related settings, such as GPIO, High Definition Audio (HAD) verb Table, HsioPtss table, SPD table. This folder also has DSC and FDF file. We can build a KabylakeRvp3 binary inside of this folder.

```
====== KabylakeOpenBoardPkg directory ======
KabylakeOpenBoardPkg
  Acpi
      BoardAcpiDxe
  FspWrapper
      Library
      PeiFspPolicyUpdateLib
  Include
  KabylakeRvp3
      Include
      Library
      OpenBoardPkg.dsc
      OpenBoardPkg.fdf
  Library
      BaseGpioExpanderLib
      PeiI2cAccessLib
  Policy
```

```
      PolicyInitDxe
====== KabylakeOpenBoardPkg directory ======
```

Summary

This section introduces the tree structure, including the platform code layout.

Feature – BIOS module selection

This section will discuss the means to enable the modules.

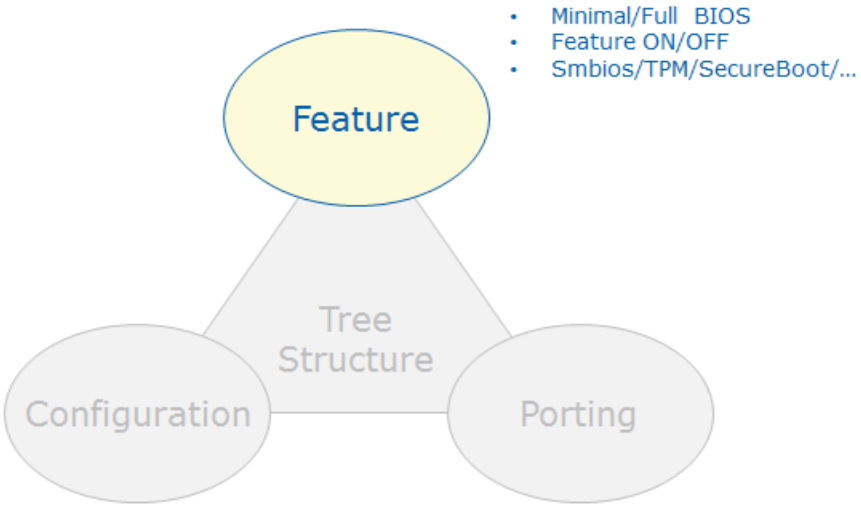

Figure 3 Focus Area - Feature

Category – minimal set versus full set

Different open source IA firmware solutions may have different feature sets. These sets are typically based upon different requirements. There are 2 possible categories:

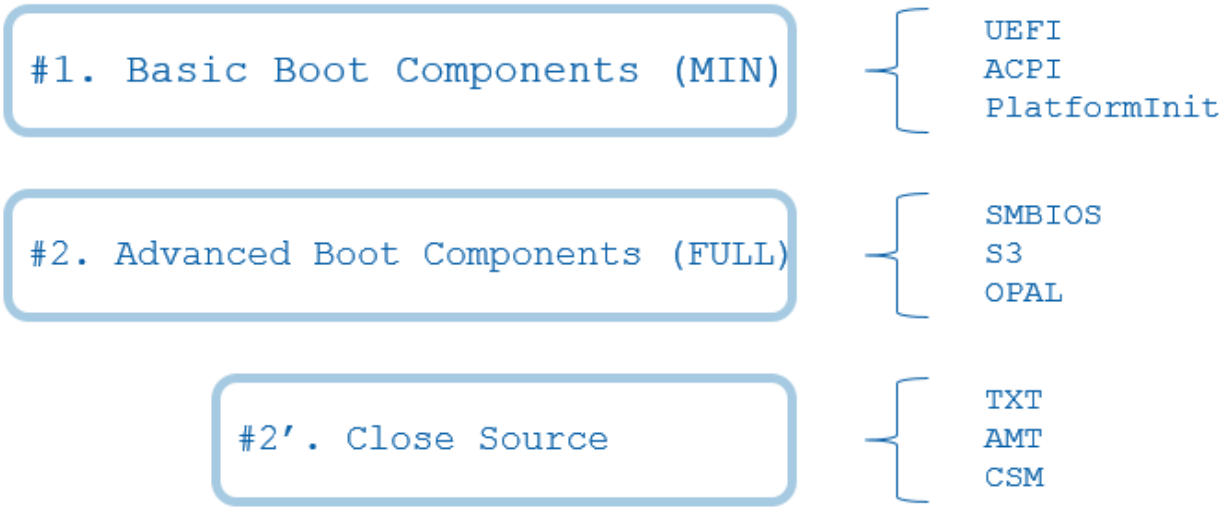

Figure 4 two categories

- Minimal set (basic boot component) – this includes the minimal components needed to boot to the UEFI Shell or to a UEFI OS. The feature set is limited and may only include a basic ACPI table and some required platform initialization.
- Full set (advanced boot component) – this entails all of the components needed to make a production BIOS. For example, it may support S3, SMBIOS table, OPAL. Most

adavenced modules can be open source, too, but there might be a small portion of code that can not be open source, such as the binary elements used by TXT/AMT/CSM.

Basic Boot Components

Per our research on the ATOM, Core-i7, and XEON platform firmware, we found the basic boot components are almost the same. In the below picture, the GREEN part means the generic EDKII core modules. The YELLOW part means the silicon specific modules. And finally, the RED part means the platform/board specific modules.

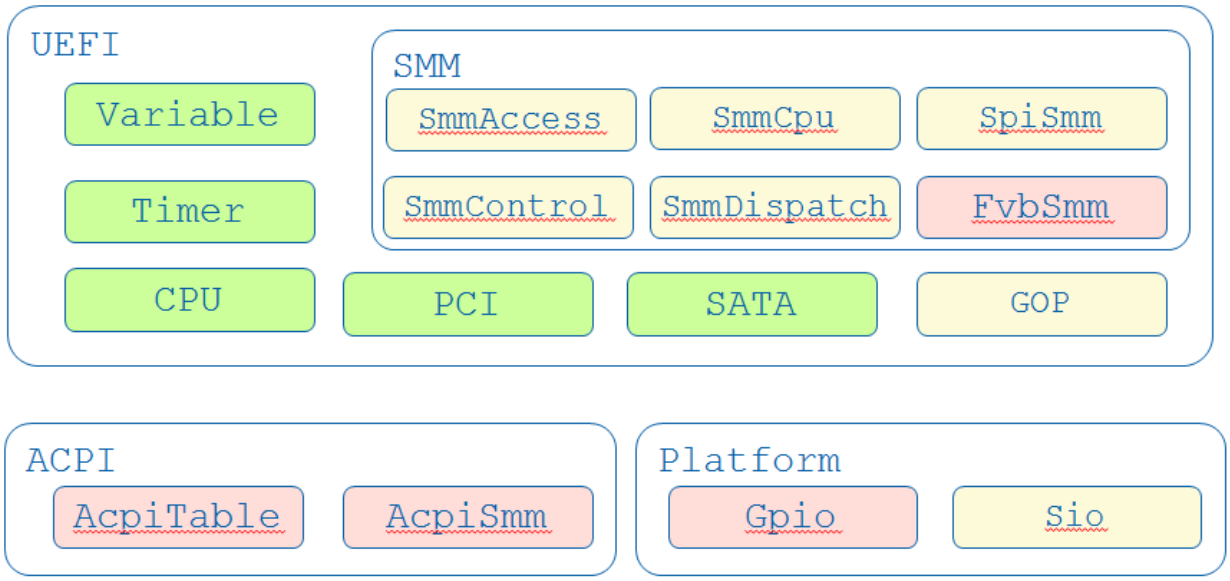

Figure 5 Basic Boot Components

In the UEFI scope, we need the variable, timer, CPU, PCI, either SATA or USB as storage, Graphic or terminal as console output, and finally, USB/PS2 Keyboard or terminal as console input. The SMM portion is required for most X86 platforms in order to support UEFI Authenticated Variables [AUTH VARIABLE].

Most UEFI OSes also require ACPI, so ACPI tables and an SMM driver to enable/disable ACPI are needed.

The platform may also need to initiliaze General Purpose Input/Ouput (GPIO) pins or a Super IO (SIO) to enable the basic boot functionality.

Guideline: Feature – table d'hôte versus à la carte

If I go to a new restaurant and do have not an idea on what to order, I will check the table d'hôte menu first. I trust the chef's recommendations.

There is a similar approach for BIOS development wherein a platform firmware infrastructure may provide a "table d'hôte" menu (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/MinPlatformPkg.dec):

```
====== Boot Stage ======
  # Stage 1 - enable debug
  # Stage 2 - mem init
  # Stage 3 - boot to shell only
  # Stage 4 - boot to OS
  # Stage 5 - boot to OS with security boot enabled
gPlatformModuleTokenSpaceGuid.PcdBootStage|4|UINT8|0xF00000A0
====== Boot Stage ======
```

This is valauble for a newcomer in order to give him or her a basic idea regarding what components are needed for a BIOS solution. It can be 3M full featured BIOS, or only 256K if just the basic boot is required, in some cases.

This work can be done by defining some default configuration in PlatformConfig.dsc.
For example, **PcdBootStage|4** can be used to configure a BIOS to support a boot to OS (with ACPI/SMM), or **PcdBootStage|3** to configure a BIOS to boot to shell only (without ACPI/SMM) (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/OpenBoardPkgConfig.dsc)

```
====== Boot Stage ======
  #
  # Please select BootStage here.
  # Stage 1 - enable debug
  # Stage 2 - mem init
  # Stage 3 - boot to shell only
  # Stage 4 - boot to OS
  # Stage 5 - boot to OS with security boot enabled
  #
  gPlatformModuleTokenSpaceGuid.PcdBootStage|4
====== Boot Stage ======
```

At the same time, a platform firmware may provide an "à la cart" menu so that an advanced user can configure an individual item. For example, **PcdUefiSecureBootEnable** can be used to configure if a BIOS needs to support UEFI secure boot [AUTH VARIABLE]. **PcdTpm2Enable** can be used to configure if a BIOS needs to support the TPM2 [TPM2 EDKII]. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/MinPlatformPkg.dec)

```
====== MinPlatformConfig.dsc ======
  gPlatformModuleTokenSpaceGuid.PcdBootToShellOnly
|FALSE|BOOLEAN|0xF00000A3
  gPlatformModuleTokenSpaceGuid.PcdUefiSecureBootEnable
|FALSE|BOOLEAN|0xF00000A4
  gPlatformModuleTokenSpaceGuid.PcdTpm2Enable
|FALSE|BOOLEAN|0xF00000A5
====== MinPlatformConfig.dsc ======
```

We recomment limiting the number of features to a reasonable level. LESS IS BETTER THAN MORE.

Feature organization

Our experience for a minimal platform is that you should have:

- ~50 core modules
- <10 silicon modules
- <10 platform modules

We observed that the platform.dsc and platform.fdf might be very long on some platforms. We recommend defining a set of dsc and fdf include files so that they can be included in a platform dsc and fdf file.

The template for MinPlatform is shown in Appendix B.
NOTE: This template is just to serve as a reference. Different platforms may choose different modules based on the platform requirements.

The common DSC (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Dsc) includes the below items:

- CoreCommonLib.dsc – The common core library.
- CorePeiLib.dsc – The core library for PEI phase.
- CoreDxeLib.dsc – The core library for DXE phase.
- CorePeiInclude.dsc – The core module built in PEI phase.
- CoreDxeInclude.dsc – The core module built in DXE phase.

The common FDF (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Fdf) includes the below items:

- CorePeiBfvInclude.fdf – The core module included in PEI BFV.
- CorePeiPreMemInclude.fdf – The core module included in PEI pre-memory FV.
- CorePeiPostMemInclude.fdf – The core module included in PEI post-memory FV.
- CoreDxeInclude.fdf – The core module included in DXE FV.
- RuleInclude.fdf – The build rule for FFS.

The project specific DSC/FDF (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3) includes the below items:

- OpenBoardPkgConfig.dsc – The configuration for the build.
- OpenBoardPkgPcd.dsc – The PCD setting for the image.
- OpenBoardPkgBuildOption.dsc – The build option for build.
- OpenBoardPkg.dsc – The main DSC to build the modules for OpenBoard.
- FlashMapInclude.fdf – The flash layout in the FD image.
- OpenBoardPkg.fdf – The main FDF to create the final FD image for OpenBoard.

Among these DSC files, OpenBoardPkgConfig.dsc is the configuration file to control whether to turn a feature on or off. For example, we can set different boot stages in this config DSC file.

The final OpenBoardPkg.dsc and OpenBoardPkg.fdf might just need to have less than ten silicon modules and less than ten platform modules. At this point, people can have a clearer picture of which modules are needed to be considered as part of any porting work.

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/OpenBoardPkg.fdf)

```
====== OpenBoardPkg.fdf ======

[FV.FVRECOVERY3]

INF $(PLATFORM_PACKAGE)/PlatformInit/PlatformInitPei/PlatformInit.inf

INF IntelFsp2WrapperPkg/FspsWrapperPeim/FspsWrapperPeim.inf

!include $(PLATFORM_PACKAGE)/Include/Build/CorePeiPostMemInclude.fdf

!if gSiPkgTokenSpaceGuid.PcdPeiDisplayEnable == TRUE
FILE FREEFORM = 4ad46122-ffeb-4a52-bfb0-518cfca02db0 {
  SECTION RAW = $(PLATFORM_FSP_BIN_PACKAGE)/SampleCode/Vbt/Vbt.bin
  SECTION UI  = "Vbt"
}
FILE FREEFORM = 7BB28B99-61BB-11D5-9A5D-0090273FC14D {
  SECTION RAW = MdeModulePkg/Logo/Logo.bmp
}
!endif # PcdPeiDisplayEnable


[FV.FVRECOVERY2]

INF $(PLATFORM_PACKAGE)/PlatformInit/PlatformInitPei/PlatformInitPreMem.inf

!include $(PLATFORM_PACKAGE)/Include/Build/CorePeiPreMemInclude.fdf

!if gPlatformModuleTokenSpaceGuid.PcdTpm2Enable == TRUE
INF $(PLATFORM_PACKAGE)/Tcg/Tcg2PlatformPei/Tcg2PlatformPei.inf
!endif

INF IntelFsp2WrapperPkg/FspmWrapperPeim/FspmWrapperPeim.inf

[FV.FVRECOVERY]


!include $(PLATFORM_PACKAGE)/Include/Build/CorePeiBfvInclude.fdf


[FV.FVMAIN]

!include $(PLATFORM_PACKAGE)/Include/Build/CoreDxeInclude.fdf

INF  $(PLATFORM_PACKAGE)/PlatformInit/PlatformInitDxe/PlatformInitDxe.inf

!if gPlatformModuleTokenSpaceGuid.PcdBootToShellOnly == FALSE
INF  $(PROJECT)/PolicyInitDxe/PolicyInitDxe.inf

$(SIPKG_DXE_SMM_BIN) $(PLATFORM_SI_PACKAGE)/Pch/PchInit/Dxe/PchInitDxe.inf
$(SIPKG_DXE_SMM_BIN)
$(PLATFORM_SI_PACKAGE)/SystemAgent/SaInit/Dxe/SaInitDxe.inf

INF  $(PLATFORM_PACKAGE)/PlatformInit/PlatformInitSmm/PlatformInitSmm.inf
$(SIPKG_DXE_SMM_BIN)
$(PLATFORM_SI_PACKAGE)/SystemAgent/SmmAccess/Dxe/SmmAccess.inf

$(SIPKG_DXE_SMM_BIN)
$(PLATFORM_SI_PACKAGE)/Pch/PchSmiDispatcher/Smm/PchSmiDispatcher.inf
```

```
$(SIPKG_DXE_SMM_BIN)
$(PLATFORM_SI_PACKAGE)/Pch/SmmControl/RuntimeDxe/SmmControl.inf
$(SIPKG_DXE_SMM_BIN) $(PLATFORM_SI_PACKAGE)/Pch/Spi/Smm/PchSpiSmm.inf
$(SIPKG_DXE_SMM_BIN) $(PLATFORM_SI_PACKAGE)/Pch/PchInit/Smm/PchInitSmm.inf

INF  $(PLATFORM_PACKAGE)/Flash/SpiFvbService/SpiFvbServiceSmm.inf

INF $(PLATFORM_PACKAGE)/Acpi/AcpiTables/AcpiPlatform.inf
INF RuleOverride = DRIVER_ACPITABLE
$(PLATFORM_BOARD_PACKAGE)/Acpi/BoardAcpiDxe/BoardAcpiDxe.inf
INF $(PLATFORM_PACKAGE)/Acpi/AcpiSmm/AcpiSmm.inf

INF  RuleOverride = ACPITABLE
$(PLATFORM_SI_PACKAGE)/SystemAgent/AcpiTables/SaAcpiTables.inf
INF  RuleOverride = ACPITABLE
$(PLATFORM_SI_PACKAGE)/SystemAgent/AcpiTables/SaSsdt/SaSsdt.inf

INF  $(PLATFORM_PACKAGE)/FspWrapper/SaveMemoryConfig/SaveMemoryConfig.inf

INF $(PLATFORM_SI_PACKAGE)/Hsti/Dxe/HstiSiliconDxe.inf

INF $(PLATFORM_PACKAGE)/Hsti/HstiIbvPlatformDxe/HstiIbvPlatformDxe.inf

!endif

INF  IntelFsp2WrapperPkg/FspWrapperNotifyDxe/FspWrapperNotifyDxe.inf

!if gPlatformModuleTokenSpaceGuid.PcdTpm2Enable == TRUE
INF $(PLATFORM_PACKAGE)/Tcg/Tcg2PlatformDxe/Tcg2PlatformDxe.inf
!endif

====== OpenBoardPkg.fdf ======
```

Summary

This section introduces the feature of BIOS module selection.

Configuration – platform policy data

This section describes platform policy data and configuration.

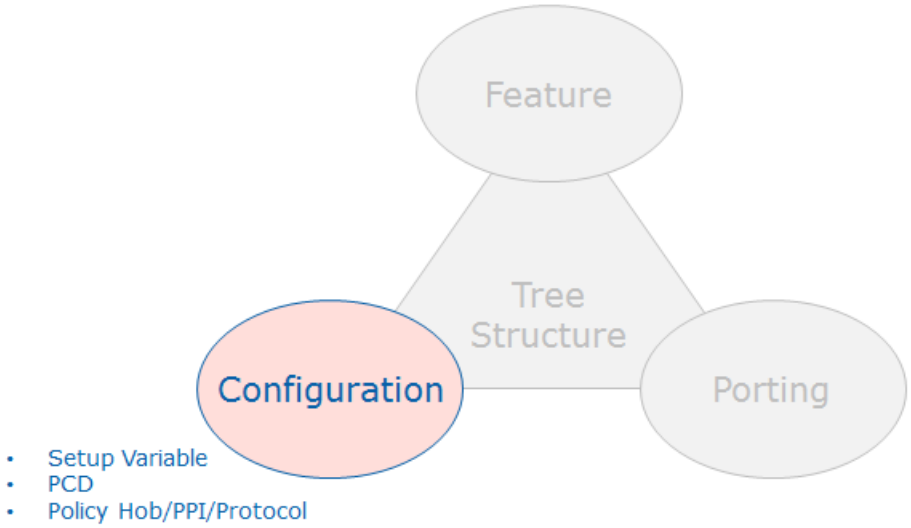

Figure 6 Focus Area – Configuration

Configuration options

As we mentioned before, there might be many sources of platform configuration data. Some general guidelines are defined below:

- **PI PCD** – The PI PCD could be static data fixed at build time or dynamic data updatable at runtime.
  - PcdsFeatureFlag: This type PCD only supports 1/0. Caller uses FeaturePcdGet() to retrieve the value. This type of PCD is mapped to be a MACRO so that a compiler optimization can remove the code scoped by "if(FALSE)". It is not allowed to set as a PcdsFeatureFlag.
  - PcdsFixedAtBuild: This type of PCD can be mapped to a global variable if the caller uses PcdGet(), or a MACRO if the caller uses FixedPcdGet(). As such, this type of PCD can be used in a data structure definition. It is not allowed to be set as PcdsFixedAtBuild.
  - PcdsPatchableInModule: This type of PCD is mapped to a global variable. It is allowed for use by both PcdGet and PcdSet. If PcdSet is called, it only changes the module-level PCD value instead of a system-level PCD value. Only the current module sees the PCD change. Other modules still see the original value.
  - PcdsDynamicDefault: PcdsDynamicDefault is mapped to a PPI or protocol. It is allowed for both PcdGet and PcdSet. PcdSet changes the system-level PCD value immediately. This type of PCD value is volatile. The changed value will not be saved in the next boot.
  - PcdsDynamicHii: PcdsDynamicHii is mapped to a UEFI variable. It is non-volatile. As such, the changed value can be saved in the next boot. However, the tricky thing is that this PCD value depends on the UEFI variable services

readiness. If PcdGet is called before UEFI variable services ready, the default PCD value will be returned instead of the updated PCD value. We suggest that the platform owner be very very careful of this trap. If DXE PcdGet is required before the UEFI variable services are ready, we suggest that the platform define PcdsDynamicDefault, and then use get variable data in the PEI phase to fill in this PCD value.

- PcdsDynamicVpd: PcdsDynamicVpd is to map configuration data to a static flash region so that a tool can modify the PcdsDynamicVpd after the flash image is generated. This is used by a BIOS that needs to support binary configuration after build. Intel FSP is an example of using PcdsDynamicVpd.
- SkuIds: SkuIds is a special usage of PCD. It can support multiple configurations generated at build time and support runtime selection to make one configuration take effect finally. The good point is that it is very straightforward for each board, if board configuration can be determined. However, current implementations just put all configuration data together without any size optimization. So even a one byte difference will cause full configurations to be duplicated. We can enhance the SkuIds PCD implementation. If there is any size concern in the SkuIds PCD, the alternative could be: define one PcdsDynamicDefault and let each platform update its own configuration there.

- **UEFI Variable** – The UEFI Variable can be non-volatile data or volatile data, and it is widely used by VFR.
  - In most cases, a non-volatile variable is used to store the user updatable configuration in a setup page. One example is VT enable/disable. This is purely a platform choice. We suggest that the platform map variable configuration to PCD, and use a PcdSet callback to set the variable data. The benefit is that if a new platform just wants to use a static setting, it can remove the variable easily.
  - A non-volatile variable may also be used to store the system configuration generated at runtime, for example, memory configuration data. In order to maintain security, we suggest that platform to lock the configuration variable before exiting PM auth/EndOfDxe event, by using the EDKII_VARIABLE_LOCK protocol.
  - A volatile variable is generated at runtime. A platform setup driver may use this information to control a VFR page to suppress or gray out a menu, or to display the system information, like CPU/SA/PCH stepping and features.
- **FSP UPD** – FSP UPD can be static default configuration, or a dynamic updatable UPD.
  - FSP UPD is used to pass configuration from the FSP wrapper into a FSP binary. A platform needs to convert the policy configuration in PCD to a FSP UPD before calling a FSP API, like FspMemoryInit, FspSiliconInit.
  - PcdsDynamicVpd.Upd: For a FSP binary, we use DynamicVpd.Upd to mark the configuration that needs to be in the UPD region. (Please be aware that UPD is not a standard PCD concept, it is an FSP extension)
- **Silicon Policy Hob/PPI/Protocol** – It is policy data constructed at runtime or it can be a hook for silicon code.
  - Policy data: Silicon Policy Hob/PPI/Protocol are useful in order to let one silicon code module support multiple boards. It is the interface between silicon code and

platform code. A platform needs to convert policy configuration in PCD into a Silicon Policy PPI/Protocol.
    - Silicon Hook: Sometimes, we observe that Silicon Policy PPI or Protocol provides a silicon hook for platform. This hook may perform some additional action based on a platform setting, or retrieve some system information. In most cases, we suggest to separate the hook function from policy data.
- **Configuration Block** – It is a data structure to put all policy data in a block without any C-language data pointer in a policy data.
    - The Configuration Block is a new idea to resolve data pointer issues objserved in earlier HOB usage. Previously, a silicon code module would define a root policy data object with some data pointers to sub-regions. For example, a PCH policy data may include a pointer to USB policy data, a pointer to SATA policy data, and a pointer to PCIExpress policy data. This HOB policy data pointer needs to be relocated after memory initialization, such as the Memory Reference Code (MRC), is done, because the PEI core needs to move data from temporary memory or Cache-As-RAM (CAR) to DRAM. If the platform code forgets to move the policy data and fix the pointer, the following code might retrieve the wrong policy data pointer. With the configuration block, the silicon and platform needn't worry about the invalid policy data pointer issue because the data pointer is eliminated. The PCH policy data block may include a USB policy data block, a SATA policy data and a PCIExpress policy data. Configuration Blocks can be mapped and used by either Hob/PPI/Protocol or PCD.
- **Global NVS** – It is an ACPI region to pass the configuration from the C code to ASL code.
    - Global NVS can be used for turning some feature on/off. An example includes returning different _STA values. 0x0 means the device does not exist. 0xF means the device exists.
    - It can be used as the policy data, for example CriticalTemperature value. A platform C code module may convert the configuration from PCD to Global NVS.
    - Since the name has "global", we observe that may platform put all different features into one big data structure. It is discouraged. We recommend each separate feature can have its own NVS data structure. It is easy for feature on/off control.
- **Platform signed data blob** – It is read only signed data at build time.
    - This signed data blob provides the configuration on a platform. An OEM may update the configuration for different boards. We suggest the platform map the signed data blob to PCDs so that a platform consumer can just use PcdGet to get the configuration without knowing the data source. The benefit is that all the platform code can be consistent, irrespective of whether the configuration data is from a signed data blob, a BIOS boot block static region, or a UEFI variable.
- **CMOS** – It is simple non-volatile storage, but it is not secure.
    - The most useful usage of CMOS is to use CMOS-CLEAR to determine if an end user wants to use the default variable configuration. This is a consistent user experience from old legacy BIOS. The new platform can use a special function key or a special GPIO as indicator of this logic.

    - The benefit of CMOS is that the CMOS can be accessed at early SEC phase without rich API requirements. Beyond that usage, though, we do not suggest a platform use CMOS to store configuration data.
- **MACRO** – C-language MACRO. It is fixed at build time.
    - A MACRO can be used as static data configuration. It is useful if the MACRO is only used in one module and does not require user configuration. However, if the MACRO is used across many modules or is configurable, like PCIE_BASE, we suggest using PCD.
    - A MACRO used in #IFDEF can be used to enable/disable features. If the consumer is in C code, it can be replaced by FeaturePcd. It will become if(0) or if(1) finally. The benefit of using PCD is that all the code in both path can be built.

Guideline: Use PCD as configuration data

PCD stands for "Platform Configuration Database". It is a platform database that contains a variety of current platform settings or directives that can be accessed by a driver or application.

We observed some platforms just access the configuration data from a special source directly, such as a UEFI variable. For the latter, people then have to remember clearly on which configuration is stored in which direct location. Also if a platform decides to change the configuration source from one place (UEFI variable) to another (static region in boot block), all impacted platform modules are required to change. This is non-ideal for source code maintenance and development.

```
====== old Platform.c======
//
// Get config from setup variable
//
VarDataSize = sizeof (SETUP_DATA);
Status = GetVariable (
                L"Setup",
                &gSetupVariableGuid,
                NULL,
                &VarDataSize,
                &mSystemConfiguration
                );

//
// Get platform info from Hob
//
HobList.Raw = GetNextGuidHob (&gPlatformInfoHobGuid, HobList.Raw);
PlatformInfo = (PLATFORM_INFO *) ((UINT8 *) (&HobList.Guid->Name) + sizeof
(EFI_GUID));
====== old Platform.c======
```

We recommend using PCD only in platform code, no matter where a platform chooses to store configuration data based upon the production requirement, like below:

```
====== new Platform.c======
//
// Get setup configuration from PCD
//
CopyMem (
```

```
    &mSystemConfiguration,
    PcdGetPtr (PcdSetupConfiguration),
    sizeof(mSystemConfiguration)
  );

  //
  // Get platform info from PCD
  //
  PlatformInfo = (PLATFORM_INFO *)PcdGetPtr (PcdPlatformInfo);
  ====== new Platform.c======
```

Then the code is consistent and easy to maintain, especially if the next generation platform decides to change the location.

Then the question becomes: how does a platform fill in configuration data that is mapped to a PCD?

We can use the below mechanism to update the existing platform code.

- **Configuration conversion**

In the current PI1.5 specification, a PCD driver can provide a callback function on PcdSet().

A platform may introduce a "ConfigConvert" module (GREEN box). The logic runs early and calls PcdSet() to convert other storage data (variable, signed data blob, policy hob) to the PCD database.

Then the rest of PlatformInit code (YELLOW box) can just call PcdGet() to get the policy data.

If the Platform driver wants to update a PCD value by calling PcdSet() later, the "ConfigConvert" can register a PCD callback function to redirect setting to other source (for example, variable).

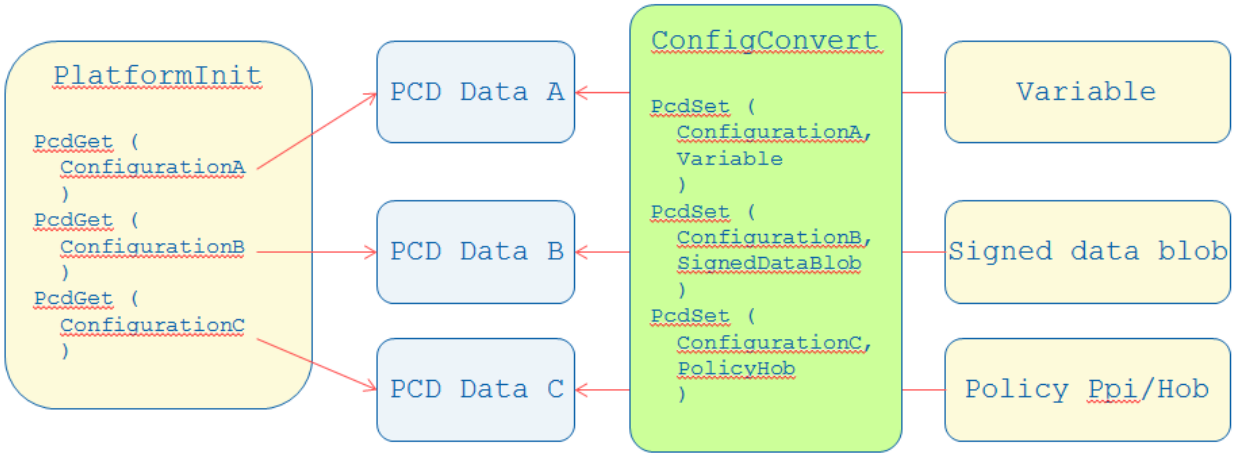

Figure 7 Configuration Conversion

The gist of configuration conversion is that any other platform driver should use PcdGet() to retrieve policy data, and PcdSet() to update policy data.

KabylakeOpenBoardPkg does not use a UEFI variable to save the configuration data, but this might be used for other real platforms.

Silicon Policy data flow

Silicon policy data update is one of the most important tasks as part of bringing up a board. In order to make the board enablement more efficient, we have the below guidelines:

- Silicon Module Provides Default Silicon Policy Data

A silicon policy data object is created per Silicon module. The data structure should only contain the data. Functions should not be used in silicon policy data.
When a silicon module installs this policy data, it should consider the most common usage as the default policy data.

- Board Module Updates the Silicon Policy Data

A board module may get default silicon policy data structures and update them.
A board module may refer to another source to get the board specific policy data, including but not limited to:
-- PCD database
-- Setup Variable
-- Binary Blob
-- Built-in C structure.

Take FSP policy in MinPlatformPkg is an example. Therein we defined FspPolicyInitLib (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/FspPolicyInitLib.h) and FspPolicyUpdateLib (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/FspPolicyUpdateLib.h). When the FSP wrapper wants to call MemoryInitAPI, it calls UpdateFspmUpdData() (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/FspWrapper/Library/PeiFspWrapperPlatformLib/PeiFspWrapperPlatformLib.c) to get the UPD configuration. Then FspmPolicyInit() and FspmPolicyUpdate() are invoked. The KabylakeSiliconPkg provides the former (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Silicon/Intel/KabylakeSiliconPkg/Library/PeiFspPolicyInitLib/PeiFspPolicyInitLib.c), and KabylakeOpenBoardPkg provides the latter (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/FspWrapper/Library/PeiFspPolicyUpdateLib/PeiFspPolicyUpdateLib.c).

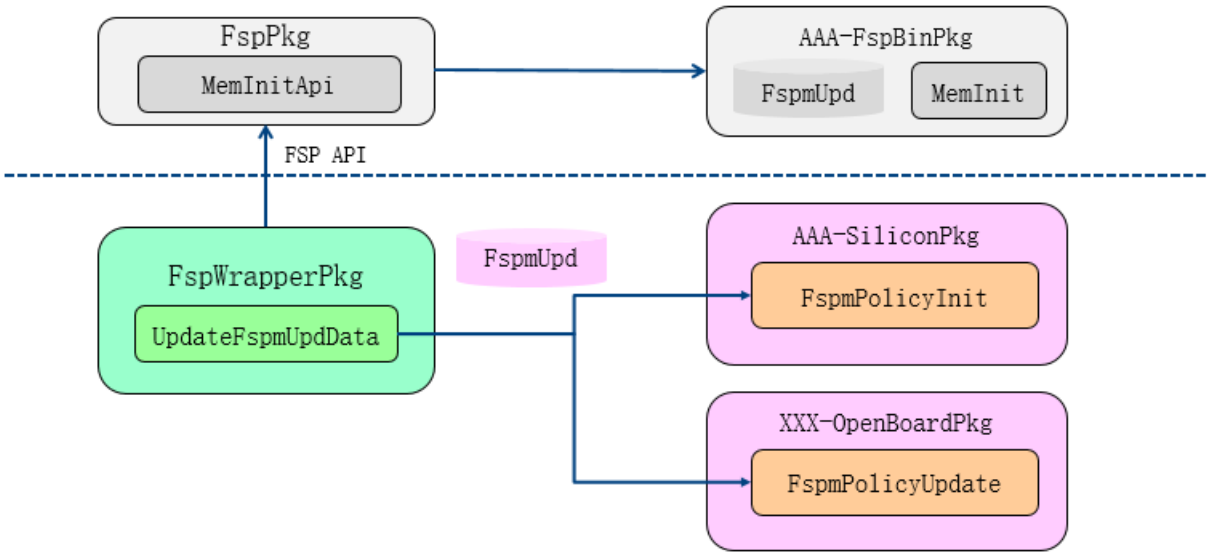

Figure 8 UPD Policy Data Flow

One example on how to update silicon policy is shown below:
(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/FspWrapper/Library/PeiFspPolicyUpdateLib/PeiSaPolicyUpdatePreMem.c)

```
====== PeiSaPolicyUpdatePreMem.c ======
EFI_STATUS
EFIAPI
PeiFspSaPolicyUpdatePreMem (
  IN OUT FSPM_UPD    *FspmUpd
  )
{
  VOID                          *Buffer;

  CopyMem((VOID *)(UINTN)FspmUpd->FspmConfig.MemorySpdPtr00, (VOID
*)(UINTN)PcdGet32 (PcdMrcSpdData), PcdGet16 (PcdMrcSpdDataSize));
  CopyMem((VOID *)(UINTN)FspmUpd->FspmConfig.MemorySpdPtr10, (VOID
*)(UINTN)PcdGet32 (PcdMrcSpdData), PcdGet16 (PcdMrcSpdDataSize));

......

  Buffer = (VOID *) (UINTN) PcdGet32 (PcdMrcRcompTarget);
  if (Buffer) {
    CopyMem ((VOID *)FspmUpd->FspmConfig.RcompTarget, Buffer, 10);
  }
  return EFI_SUCCESS;
}
====== PeiSaPolicyUpdatePreMem.c ======
```

Summary
This section introduces configuration – platform policy data.

Porting – Board Specific initialization

This section talks about board porting.

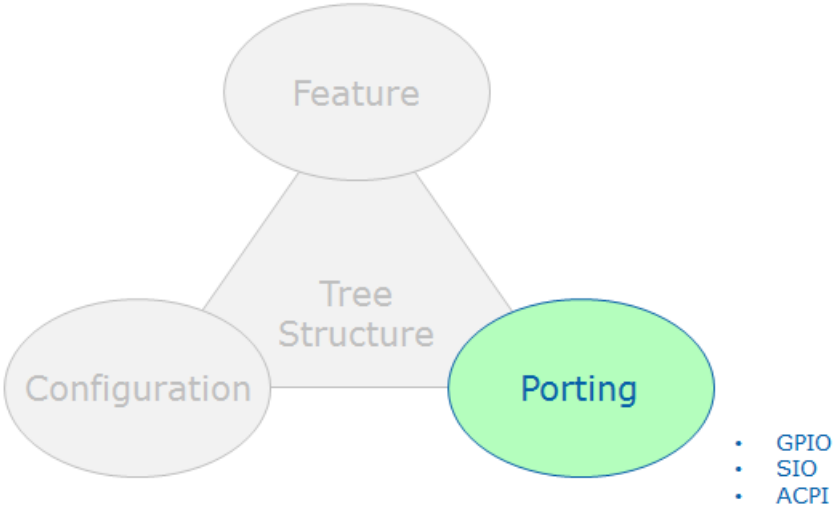

Figure 9 Focus Area – Porting

Platform Initialization Flow

The PlatformInit folder (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/PlatformInit) - PlatformInitPei, PlatformInitDxe and PlatformInitSmm control the platform initialization flow. Because this flow needs to involve the board initialization, we define a set of board hook points in BoardInitLib (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/BoardInitLib.h)

Table 1 Board init hook point

Phase	Function	Comment
SEC	BoardBeforeTempRamInit()	No Stack
	BoardAfterTempRamInit()	Stack is setup.
PEI	BoardDetect()	Variable service is ready.
	BoardDebugInit()	
	BoardBootModeDetect()	
	BoardInitBeforeMemoryInit()	
	BoardInitBeforeTempRamExit()	
	BoardInitAfterTempRamExit()	
	BoardInitAfterMemoryInit()	Memory Discover Callback
	BoardInitBeforeSiliconInit()	
	BoardInitAfterSiliconInit()	EndOfDxe Callback
DXE	BoardInitAfterPciEnumeration()	
	BoardInitReadyToBoot()	

	BoardInitEndOfFirmware()	

Take the below PlatformInitPei (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/PlatformInit/PlatformInitPei) as an example. The YELLOW highlighted function is the board init hook function. The BLUE highlighted function is the test point check function, which will be discussed in the later Testibility chapter. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/PlatformInit/PlatformInitPei/PlatformInitPreMem.c)

```
====== PlatformInitPreMem.c ======
EFI_STATUS
EFIAPI
PlatformInitPreMem (
  IN CONST EFI_PEI_SERVICES     **PeiServices,
  IN EFI_PEI_NOTIFY_DESCRIPTOR  *NotifyDescriptor,
  IN VOID                       *Ppi
  )
{
  BoardDetect ();

  BoardDebugInit ();

  TestPointDebugFunction ();

  BootMode = BoardBootModeDetect ();
  Status = PeiServicesSetBootMode (BootMode);
  if (BootMode == BOOT_IN_RECOVERY_MODE) {
    Status = PeiServicesInstallPpi (&mPpiListRecoveryBootMode);
  }
  Status = PeiServicesInstallPpi (&mPpiBootMode);

  BuildMemoryTypeInformation ();

  Status = BoardInitBeforeMemoryInit ();
}

EFI_STATUS
EFIAPI
MemoryDiscoveredPpiNotifyCallback (
  IN CONST EFI_PEI_SERVICES     **PeiServices,
  IN EFI_PEI_NOTIFY_DESCRIPTOR  *NotifyDescriptor,
  IN VOID                       *Ppi
  )
{
  Status = BoardInitAfterMemoryInit ();

  SetCacheMtrr ();

  ReportCpuHob ();

  ///
  /// If S3 resume, then we are done
  ///
  if (BootMode == BOOT_ON_S3_RESUME) {
    return EFI_SUCCESS;
  }

  ReportFv ();
```

```
  TestPointMemoryDiscovered ();
}

====== PlatformInitPreMem.c ======
```

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/PlatformInit/PlatformInitPei/PlatformInitPostMem.c)

```
====== PlatformInitPostMem.c ======
EFI_STATUS
EFIAPI
PlatformInitPostMemEntryPoint (
  IN       EFI_PEI_FILE_HANDLE  FileHandle,
  IN CONST EFI_PEI_SERVICES     **PeiServices
  )
{
  Status = BoardInitBeforeSiliconInit ();
}

EFI_STATUS
EFIAPI
PlatformInitEndOfPei (
  IN CONST EFI_PEI_SERVICES     **PeiServices,
  IN EFI_PEI_NOTIFY_DESCRIPTOR  *NotifyDescriptor,
  IN VOID                       *Ppi
  )
{
  Status = BoardInitAfterSiliconInit ();

  Status = SetCacheMtrrAfterEndOfPei ();

  TestPointEndOfPei ();
}
====== PlatformInitPostMem.c ======
```

The board code just needs to implement the required function, an it can be included into the platform initialization process. For example, all the KabylakeRvp3 board init code is at https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib.

The BoardInitLib instance is at https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiBoardInitPreMemLib.c and https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiBoardInitPostMemLib.c

Multi-board support

There might be a requirement that one BIOS should be able to boot multiple boards. Different boards may have different configurations.

We observed many BIOS code examples that use switch-case style board ID checks in many platform drivers.
For example:

```
====== old PlatformGpio.c======
switch (BoardId) {
case BoardIdBoard1:
    GpioPin = GPIO_BOARD1;
    Break;
case BoardIdBoard2:
    GpioPin = GPIO_BOARD2;
    Break;
case BoardIdBoard3:
    GpioPin = GPIO_BOARD3;
    Break;
}
====== old PlatformGpio.c======

====== old PlatformRecovery.c======
switch (BoardId) {
case BoardIdBoard1:
    IsRecovery = IsRecoveryBoard1 ();
    Break;
case BoardIdBoard2:
    IsRecovery = IsRecoveryBoard2 ();
    Break;
case BoardIdBoard3:
    IsRecovery = IsRecoveryBoard3 ();
    Break;
}
====== old PlatformRecovery.c======

====== old PlatformAcpi.c======
switch (BoardId) {
case BoardIdBoard1:
    AcpiConfig = ACPI_BOARD1;
    Break;
case BoardIdBoard2:
    AcpiConfig = ACPI_BOARD2;
    Break;
case BoardIdBoard3:
    AcpiConfig = ACPI_BOARD3;
    Break;
}
====== old PlatformAcpi.c======
```

A real example in Quark:
https://github.com/tianocore/edk2/blob/master/QuarkPlatformPkg/Platform/Pei/PlatformInit/PlatformEarlyInit.c

```
====== Quark PlatformEarlyInit.c======
/**
  Initialize state of I2C GPIO expanders.

  @param  PlatformType  Platform type for GPIO expander init.

**/
EFI_STATUS
EarlyPlatformConfigGpioExpanders (
  IN CONST EFI_PLATFORM_TYPE              PlatformType
  )
{
  ......
  if (PlatformType == GalileoGen2) {
    //
    // Configure AMUX1_IN (EXP2.P1_4) as an output
    //
    PlatformPcal9555GpioSetDir (
```

```
      GALILEO_GEN2_IOEXP2_7BIT_SLAVE_ADDR,  // IO Expander 2.
      12,                                   // P1-4.
      FALSE                                 // Configure as output
      );
    ......
  }

  if (PlatformType == Galileo) {
    //
    // Detect the I2C Slave Address of the GPIO Expander
    //
    if (PlatformLegacyGpioGetLevel (R_QNC_GPIO_RGLVL_RESUME_WELL,
GALILEO_DETERMINE_IOEXP_SLA_RESUMEWELL_GPIO)) {
      I2CSlaveAddress.I2CDeviceAddress = GALILEO_IOEXP_J2HI_7BIT_SLAVE_ADDR;
    } else {
      I2CSlaveAddress.I2CDeviceAddress = GALILEO_IOEXP_J2LO_7BIT_SLAVE_ADDR;
    }
    ......
  }

}
====== Quark PlatformEarlyInit.c======
```

Similar code in https://github.com/tianocore/edk2/blob/master/QuarkPlatformPkg/Library/PlatformPcieHelperLib/PlatformPcieHelperLib.c and https://github.com/tianocore/edk2/blob/master/QuarkPlatformPkg/Library/PlatformSecureLib/PlatformSecureLib.c

The problem is that if a developer wants to add a new board, he/she must go through all the code to find out what needs to be changed in each driver. It is a huge burden.

Guideline: One board, one directory

We recommend creating a directory for each board and put all board specific settings in this board directory. For example, we put KabylakeRvp3 (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3) as a folder in KabylakeOpenBoardPkg.

```
====== KabylakeOpenBoardPkg directory ======
KabylakeOpenBoardPkg
  Acpi
  FspWrapper
      Library
          PeiFspPolicyUpdateLib
  Include
  Library
  KabylakeRvp3
      Include
      Library
          BasePlatformHookLib
          BoardAcpiLib
          BoardInitLib
      OpenBoardPkg.dsc
      OpenBoardPkg.fdf
====== KabylakeOpenBoardPkg directory ======
```

If we need to add a new board, such as Rvp7, we can copy the KabylakeRv3 folder to KabylakeRvp7 folder, and update all the modules in this KabylakeRvp7.

Once we move the board specific code to the board specific directory, the generic board code should not contain any board specific code. For example, we do not put PeiFspPolicyUpdateLib (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/FspWrapper/Library/PeiFspPolicyUpdateLib) into KabylakeRvp3, because this code only consumes a set of PCDs, such as PcdMrcRcompResistor, PcdSpecificLpHsioPtssTable1, PcdHdaVerbTable, etc. A KabylakeRvp3 board specific code (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib) produces these PCDs and Kabylake common board code consumes these PCDs.

Board detection

In order to detemine which board specific driver needs to run and which does not need to run, there must be some code to detect the board type.
The board detection code is board specific. It should be under the board specific directory.

We define a BoardDetect() API in BoardInitLib (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/BoardInitLib.h) and a RegisterBoardDetect() API in MultiBoardInitSupportLib (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/MultiBoardInitSupportLib.h).

If the final BIOS image only needs to support one board, the board code can just implement the BoardDetect() API directly. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiBoardInitPreMemLib.c)

```
====== PeiBoardInitPreMemLib.c ======
EFI_STATUS
EFIAPI
BoardDetect (
  VOID
  )
{
  KabylakeRvp3BoardDetect ();
  return EFI_SUCCESS;
}
====== PeiBoardInitPreMemLib.c ======
```

If the final BIOS image needs to support multiple boards, the board code needs to implement an AAABoardDetect() and use RegisterBoardDetect() API to register such a function in a BoardInit module entrypoint or library constructor. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiMultiBoardInitPreMemLib.c)

```
====== PeiMultiBoardInitPreMemLib.c ======
BOARD_DETECT_FUNC  mKabylakeRvp3BoardDetectFunc = {
  KabylakeRvp3MultiBoardDetect
};
```

```
EFI_STATUS
EFIAPI
KabylakeRvp3MultiBoardDetect (
  VOID
  )
{
  KabylakeRvp3BoardDetect ();
  if (LibPcdGetSku () == BoardIdKabyLakeYLpddr3Rvp3) {
    RegisterBoardPreMemInit (&mKabylakeRvp3BoardPreMemInitFunc);
  }
  return EFI_SUCCESS;
}

EFI_STATUS
EFIAPI
PeiKabylakeRvp3MultiBoardInitPreMemLibConstructor (
  VOID
  )
{
  return RegisterBoardDetect (&mKabylakeRvp3BoardDetectFunc);
}
====== PeiMultiBoardInitPreMemLib.c ======
```

When the PlatformInitPei module runs, it calls BoardDetect(). If the BIOS image supports one board, this code calls into board specific API directly. If the BIOS image support multiple board, this code calls into a generic MultiBoardInitSupportLib/BoardInitLib instance (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/PlatformInit/Library/MultiBoardInitSupportLib/PeiBoardInitLib.c), and all the registered BoardDetec functions are invoked one by one.

Once a board detection function successfully recognizes the board, it can set SkuId to indicate the board detection is done. This SkuId is used as inicator that – board detection is successful and is finished, so that any other board detection function will just return immediately.

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiKabylakeRvp3Detect.c)

```
====== PeiKabylakeRvp3Detect.c ======
EFI_STATUS
EFIAPI
KabylakeRvp3BoardDetect (
  VOID
  )
{
  if (LibPcdGetSku () != 0) {
    return EFI_SUCCESS;
  }

  DEBUG ((EFI_D_INFO, "KabylakeRvp3DetectionCallback\n"));

  if (IsKabylakeRvp3 ()) {
    LibPcdSetSku (BoardIdKabyLakeYLpddr3Rvp3);

    DEBUG ((DEBUG_INFO, "SKU_ID: 0x%x\n", LibPcdGetSku()));
    ASSERT (LibPcdGetSku() == BoardIdKabyLakeYLpddr3Rvp3);
  }
  return EFI_SUCCESS;
```

```
}
====== PeiKabylakeRvp3Detect.c ======
```

Board initialization

Once Board detection is successful, the next step is Board initialization.

If the final BIOS image only needs to support one board, the board code can just implement a set of Board initialization API's directly. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiBoardInitPreMemLib.c)

```
====== PeiBoardInitPreMemLib.c ======
EFI_STATUS
EFIAPI
BoardDebugInit (
  VOID
  )
{
  KabylakeRvp3BoardDebugInit ();
  return EFI_SUCCESS;
}

EFI_BOOT_MODE
EFIAPI
BoardBootModeDetect (
  VOID
  )
{
  return KabylakeRvp3BoardBootModeDetect ();
}

EFI_STATUS
EFIAPI
BoardInitBeforeMemoryInit (
  VOID
  )
{
  KabylakeRvp3BoardInitBeforeMemoryInit ();
  return EFI_SUCCESS;
}

====== PeiBoardInitPreMemLib.c ======
```

If the final BIOS image needs to support multiple boards, the board detection code needs to check the SkuId PCD to decide if the current board is detected. Then the same function uses RegisterBoardPreMemInit() API to register a set of PreMem initialization functions in BoardDetect. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiMultiBoardInitPreMemLib.c)

```
====== PeiMultiBoardInitPreMemLib.c ======

BOARD_PRE_MEM_INIT_FUNC  mKabylakeRvp3BoardPreMemInitFunc = {
  KabylakeRvp3BoardDebugInit,
  KabylakeRvp3BoardBootModeDetect,
  KabylakeRvp3BoardInitBeforeMemoryInit,
  NULL, // BoardInitAfterMemoryInit
```

```
  NULL, // BoardInitBeforeTempRamExit
  NULL, // BoardInitAfterTempRamExit
};

EFI_STATUS
EFIAPI
KabylakeRvp3MultiBoardDetect (
  VOID
  )
{
  KabylakeRvp3BoardDetect ();
  if (LibPcdGetSku () == BoardIdKabyLakeYLpddr3Rvp3) {
    RegisterBoardPreMemInit (&mKabylakeRvp3BoardPreMemInitFunc);
  }
  return EFI_SUCCESS;
}
====== PeiMultiBoardInitPreMemLib.c ======
```

The board specific initialization may include:
1) Invoke a set of PCD for policy initialization later.
2) Configure the hardware devices (such as GPIO, SIO)

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiKabylakeRvp3InitPreMemLib.c)

```
====== PeiKabylakeRvp3InitPreMemLib.c ======
EFI_STATUS
EFIAPI
KabylakeRvp3InitPreMem (
  VOID
  )
{
  PcdSet32S (PcdPcie0WakeGpioNo, 0);
  PcdSet8S  (PcdPcie0HoldRstExpanderNo, 0);
  PcdSet32S (PcdPcie0HoldRstGpioNo, 8);

  PcdSet32S (PcdMrcRcompResistor, (UINTN) RcompResistorSklRvp1);
  PcdSet32S (PcdMrcRcompTarget, (UINTN) RcompTargetSklRvp1);

  ......
  PcdSet32S (PcdMrcSpdData, (UINTN) mSkylakeRvp3Spd110);
  PcdSet16S (PcdMrcSpdDataSize, mSkylakeRvp3Spd110Size);

  PcdSetBoolS (PcdIoExpanderPresent, TRUE);

  return EFI_SUCCESS;
}

EFI_STATUS
EFIAPI
KabylakeRvp3BoardInitBeforeMemoryInit (
  VOID
  )
{
  KabylakeRvp3InitPreMem ();

  //
  // Configures the I2CGpioExpander
  //
  if (PcdGetBool (PcdIoExpanderPresent)) {
```

```
    I2CGpioExpanderInitPreMem();
  }

  GpioInitPreMem ();
  SioInit ();

  ///
  /// Do basic PCH init
  ///
  SiliconInit ();

  return EFI_SUCCESS;
}
====== PeiKabylakeRvp3InitPreMemLib.c ======
```

The PCD data/data pointer is used for silicon FSP policy initialization.

Board specific module

If the final BIOS image needs to support multi board, more than one board directory may be involved. There might be a set of drivers under the BoardAAA directory, a set of drivers under the BoardBBB directory. How do we know which one will run and take effect finally?

```
====== XXXOpenBoardPkg directory ======
XXXOpenBoardPkg
  BoardAAA
      Library
      DriverXXX
  BoardBBB
      Library
      DriverYYY
====== XXXOpenBoardPkg directory ======
```

- **SKUID check**

The board specific module entry point may add a board ID check and run if and only if the board ID matches.

```
====== BoardFeature.c======
EFI_STATUS
InstallBoardFeature (
  IN EFI_HANDLE         ImageHandle,
  IN EFI_SYSTEM_TABLE   *SystemTable
  )
{
  EFI_STATUS                    Status;

  if (LibPcdGetSku () == BoardIdKabyLakeYLpddr3Rvp3) {
    return EFI_UNSUPPORTED;
  }
  //
  // Do initialization
  //
  ......
}
=====================
```

NOTE: The SKU ID check should only happen in board specific drivers. The SKU ID check is NOT allowed in any common board code or common platform code.

Board specific ACPI

- **Board specific device selection**

We observed many BIOS code modules define BoardId in ACPI global NVS and use the board ID check in the ASL code. For example:

```
====== Old asl ======
  Device(DEV0)
  {
    Method(_STA,0)
    {
      If(LEqual(BID,BoardIdBoardX)
      {
        Return(0x0000)
      }
      Return(0x001F)
    }
====== Old asl ======
```

This approach is NOT portable if we want to add a new board.

One way to resolve this issue is to define a board-neutral name for the branch condition. For example: DEVP means Device Present.

```
====== recommended asl ======
  Device(DEV0)
  {
    Method(_STA,0)
    {
      If(LEqual(DEVP,0)
      {
        Return(0x0000)
      }
      Return(0x001F)
    }
====== recommended asl ======
```

So that in the board specific AcpiBoard.c, each board can configure this value.

```
====== Recommanded AcpiBoard ======
VOID
InstallAcpiBoard (
  VOID
  )
{
  ......
  mGlobalNvsArea.Area->DEVP = 0;

}
====== Recommanded AcpiBoard ======
```

A real example in KabylakeRvp3 is below (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardAcpiLib/DxeKabylakeRvp3AcpiTableLib.c)

```
====== DxeKabylakeRvp3AcpiTableLib.c ======
VOID
KabylakeRvp3UpdateGlobalNvs (
  VOID
```

```
  )
{
  mGlobalNvsArea.Area = (VOID *)(UINTN)PcdGet64 (PcdAcpiGnvsAddress);

  mGlobalNvsArea.Area->PowerState = 1;
  mGlobalNvsArea.Area->NativePCIESupport = PcdGet8 (PcdPciExpNative);
  mGlobalNvsArea.Area->ApicEnable = GLOBAL_NVS_DEVICE_ENABLE;
  mGlobalNvsArea.Area->LowPowerS0Idle = PcdGet8 (PcdLowPowerS0Idle);
  mGlobalNvsArea.Area->Ps2MouseEnable = FALSE;
  mGlobalNvsArea.Area->Ps2KbMsEnable  = PcdGet8 (PcdPs2KbMsEnable);
}
====== DxeKabylakeRvp3AcpiTableLib.c ======
```

If this device is a silicon device and it might be enabled/disabled by policy, we recommend using the above mechanism.

If this device is a board device and it exists only on boardX but does not exist on boardY, we recommend using the below mechanism – SSDT.

- **Board specific SSDT**

The other way to resolve above issue is to move the board specific ACPI Secondary System Description Table (SSDT) to the board specific directory and let it be installed by a board specific ACPI driver. The basic platform ACPI driver should only handle generic ACPI tables, like FADT, MCFG, HPET, MCFG, and etc.

The old QuarkPlatform merged all device specific ASL code together into one big DSDT (AD7298/ CY8C9540A/ GpioClient/ ADC108S102/ CAT24C08/ PCA9685/ PCAL9555A) and used PlatformType to check if the device exists or not. (https://github.com/tianocore/edk2/blob/master/QuarkPlatformPkg/Acpi/AcpiTables/Dsdt/PCAL9555A.asi)

```
====== old PCAL9555A.asi ======
Device(NIO1)
{
  ......
    Method(_STA, 0x0, NotSerialized)
    {
        //
        // Only Platform Type / Id 8 has this device.
        //
        If(LNotEqual(PTYP, 8))
        {
          return (0)
        }
        Return(0xf)
    }
}
====== old PCAL9555A.asi ======
```

This code has same problem as PlatformInit: if a developer wants to add a new board, or wants to reuse the current device, he/she must go through all the code to find out what need to be changed in each driver. It is huge burden.

A better way is to keep those board specific SSDT in board driectly.

```
====== XXXOpenBoardPkg directory ======
XXXOpenBoardPkg
  BoardAAA
      Library
      BoardAcpiSsdt
  BoardBBB
      Library
      BoardAcpiSsdt
====== XXXOpenBoardPkg directory ======
```

- **ACPI Global NVS usage**

We also observed some BIOS code defines a huge amount of board specific configuration in the global NVS area. This is not good approach because a generic platform should not have the board specific knowledge.

For example:
The old QuarkPlatform has `AlternateSla` in global NVS area, but it is Galileo board specific: (https://github.com/tianocore/edk2/blob/master/QuarkPlatformPkg/Acpi/Dxe/AcpiPlatform/AcpiPlatform.c)

```
====== old AcpiPlatform.c ======
EFI_STATUS
AcpiPlatformEntryPoint (
  IN EFI_HANDLE     ImageHandle,
  IN EFI_SYSTEM_TABLE   *SystemTable
  )
{
  ......
  if (mGlobalNvsArea.Area->PlatformType == Galileo) {
    if (PlatformLegacyGpioGetLevel (R_QNC_GPIO_RGLVL_RESUME_WELL,
GALILEO_DETERMINE_IOEXP_SLA_RESUMEWELL_GPIO)) {
      mGlobalNvsArea.Area->AlternateSla = FALSE;
    } else {
      mGlobalNvsArea.Area->AlternateSla = TRUE;
    }
  }
  ......
}
====== old AcpiPlatform.c ======
```

The recommend way is:

1) If this Global NVS is for a board specific device, it should be moved to the board specific directory. A board should define a board specific NVS, or just use a simple Name object under this device node.
2) If this Global NVS is for a board advanced feature, it should be moved to the feature directory. This fetaure driver should define a feature specific NVS, or just use a simple Name object for the feature.
3) The Global NVS ASL definition (such as https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/Include/Acpi/GlobalNvs.asl) and C-language definition (such as https://github.com/tianocore/edk2-platforms/blob/devel-

MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/Include/Acpi/GlobalNvsAreaDef.h) should be put to same directory. As such people can compare them to know the mapping between C-language definition and ASL definition. A better is to use a tool to create one from the other to avoid mismatch issue.

Board specific VFR

We observed many BIOS code modules use the board ID check in VFR code. For example:

```
====== Old vfr ======
 oneof varid = SETUP_DATA.CameraType,
       prompt      = STRING_TOKEN(STR_CAMERA_TYPE),
       help        = STRING_TOKEN(STR_CAMERA_TYPE_HELP),
       default value = cond(ideqvallist SETUP_VOLATILE_DATA.PlatId ==
BoardIdBoardX ? 0x0:0x1), defaultstore = MyStandardDefault,
       option text = STRING_TOKEN(STR_IVCAM_CAMERA), value = 0, flags = DEFAULT
| MANUFACTURING | RESET_REQUIRED;
       option text = STRING_TOKEN(STR_DS4_CAMERA),   value = 1, flags =
RESET_REQUIRED;
 endoneof;
====== Old vfr ======
```

This “default value” keyword is good for VFR because it provides a flexible way to determine a default value. However it is not good way for a board ID check. It is NOT portable if we want to add a new board. The better way is to add board specific override in BoardInit function. The override may happen in BoardInitBeforeMemoryInit() or BoardInitBeforeSiliconInit(), based upon the need.

```
====== recommended vfr ======
       prompt      = STRING_TOKEN(STR_CAMERA_TYPE),
       help        = STRING_TOKEN(STR_CAMERA_TYPE_HELP),
       option text = STRING_TOKEN(STR_IVCAM_CAMERA), value = 0, flags = DEFAULT
| MANUFACTURING | RESET_REQUIRED;
       option text = STRING_TOKEN(STR_DS4_CAMERA),   value = 1, flags =
RESET_REQUIRED;
 endoneof;
====== recommended vfr ======
```

- **Board default setup variable data**

If the BIOS uses different default setup variable values for different boards, some additional steps are needed. During the BIOS build phase, a FCE (Firmware Configuration Edit) tool (https://firmware.intel.com/sites/default/files/2015-WW48-FCE.31.zip) scans BIOS binary, extracts the setup IFR binary, and saves a set of configuration to the BIOS boot block. For example, if a BIOS supports 3 boards (Board1, Board2, Board3), there are 3 configuration data objects (Board1Config, Board2Config, Board3Config) saved in the BIOS boot block.

Because we do not recommend using board ID in the VFR page, we eliminate the multi board configuration. However the concept of default configuration is very important because it provides the system a way to recover if the user configuration causes system crash. How do we support that?

During BIOS boot time, the board initialization module calls BoardXConfigInit() to initialize the configuration data. As a default policy, the configuration data is stored in the UEFI variable region or a special PCD (PcdNvStorageDefault). If all board configuration is set by SKU PCD, then no addition step is needed. But if there is some field need update at runtime, we need CreateNvStorageDefault() to update the PcdNvStorageDefault in the C code.

```
====== BoardConfigDefault.c ======
EFI_STATUS
CreateNvStorageDefault (
  VOID
  )
{
  //
  // Get board neutral config, and override it with Board specific data.
  // This step is not needed, if all config data in SKU PCD.
  //
  NvStorageDefault = PcdGetPtr (PcdNvStorageDefault);
  NvStorageDefault->Abc = 0x123;
  PcdSetPtrS (PcdNvStorageDefault, &DataSize, NvStorageDefault);
}
====== BoardConfigDefault.c ======
```

In some special cases, like CMOS-CLEAR, boot with default, recovery mode, or no variable because of first boot, the BIOS needs to use default configuration data. A board code need have some special check to decide if default configuration is needed in this boot, like below:

```
====== BoardConfigDefaultCheck.c ======
BOOLEAN
NeedDefaultConfig (
  VOID
  )
{
  //
  // Check CMOS battery is ok.
  //
  if (IsCmosBad ()) {
    DEBUG ((DEBUG_INFO, "CMOS battery is bad. Reset the Setup variable.\n"));
    return TRUE;
  }

  //
  // Check BootMode on Recovery boot or Boot with Default settings.
  //
  else if (BootMode == BOOT_IN_RECOVERY_MODE || BootMode ==
BOOT_WITH_DEFAULT_SETTINGS) {
    return TRUE;
  }

  //
  // Check whether Setup Variable does exist to know the first boot or not.
  //
  Status = PeiGetVariable (L"Config", &gConfigVarGuid, &Attributes, &DataSize,
&Config);
  if (Status == EFI_NOT_FOUND) {
    return TRUE;
  }

  return FALSE;
```

```
}
====== BoardConfigDefaultCheck.c ======
```

Finally, the board code needs another special PCD (PcdNvStorageCurrent) to record the configuration for current boot, besides PcdNvStorageDefault as the default value. The PcdNvStorageCurrent might be updated based upon the end user action, such as BIOS setup update. The PcdNvStorageDefault is not updated by the end user. Below code shows how PcdNvStorageCurrent is set.

```
====== BoardConfigCurrent.c ======
EFI_STATUS
CreateNvStorageCurrent (
  VOID
  )
{
  if (NeedDefaultConfig ()) {
    //
    // Use default config as current config
    //
    NvStorageDefault = PcdGetPtr(PcdNvStorageDefault);
    PcdSetPtrS (PcdNvStorageCurrent, &DataSize, PcdGetPtr(PcdNvStorageDefault));

    //
    // Create variable hob, so that the config will be save to NV in DXE.
    //
    CreateVariableHob ();
    //
    // This step is not needed, if all config data in SKU PCD.
    //
    SetVariableToHob (L"Config", &gConfigVariableGuid, Attributes, DataSize,
NvStorageDefault);

  } else {
    //
    // Just use variable data as current config.
    //
    PeiGetVariable (L"Setup", &gSetupVariableGuid, &Setup, &DataSize);
    PcdSetPtrS (PcdNvStorageCurrent, &DataSize, &Setup);
  }
}
====== BoardConfigCurrent.c ======
```

If default configuration is used, PcdNvStorageCurrent is synced from PcdNvStorageDefault. At same time, we need create the VariableHob to override the variable data on the flash region. This variable hob will be synced to non-volatile memory by the VariableRuntimeDxe driver later.

If no default configuration is needed, PcdNvStorageCurrent is synced from the setup variable directly. There is no need to update PcdNvStorageDefault or create variable hob.

A full solution need create both PcdNvStorageDefault and PcdNvStorageCurrent, because the PcdNvStorageDefault will be used in setup driver later. We will discuss that in next section.

```
====== BoardConfig.c ======
EFI_STATUS
BoardXConfigInit (
  VOID
  )
{
```

```
  CreateNvStorageDefault ();

  CreateNvStorageCurrent ();
}

====== BoardConfig.c ======
```

The below figure shows the flow:

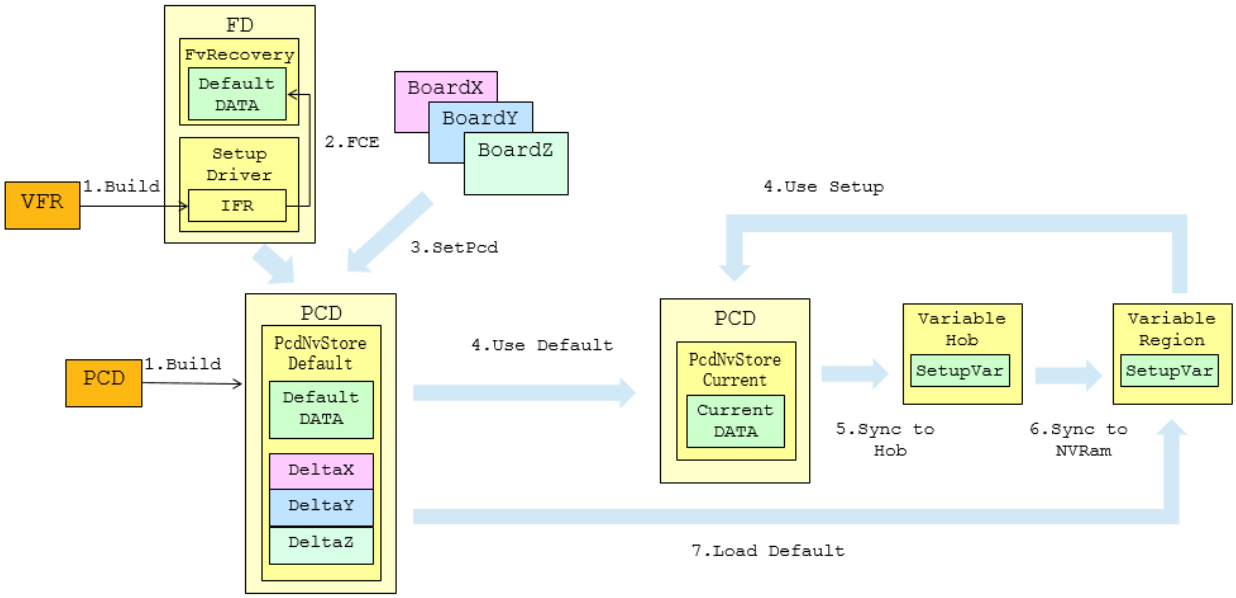

Figure 10 Multi-board support for setup configuration

- **Setup LoadDefault support**

Sometimes a setup driver can have a feature to load a default configuration based upon if the user updates some configuration by mistake. This default configuration might be different on each board.

The PCD PcdNvStorageCurrent always contains the *current* setting, but not the *default* setting.

The DefaultConfig inserted by the FCE tool is only *common* default values, but not board specific default values.

In order to support “LoadDefault”, we need another PcdNvStorageDefault. The PCD PcdNvStorageDefault contains board specific *default* settings. It should also be updated by BoardXConfigInit() according to different board types.

Then when the setup driver gets a “LoadDefault” request later, the setup driver uses the value in PcdNvStorageDefault.

SKU-PCD

SkuIds is a special usage of PCD. It can support multiple configurations generated at build time, and it supports runtime selection to make one configuration take effect finally. Below is an example that shows how to use SKU-PCD.

```
====== dsc file ======
[SkuIds]
  0|DEFAULT              # The entry: 0|DEFAULT is reserved and always required.
  4|BoardX
  0x42|BoardY

[PcdsDynamicDefault.common.BoardX]
    gBoardModuleTokenSpaceGuid.PcdGpioPin|0x8
    gBoardModuleTokenSpaceGuid.PcdGpioInitValue|{0x00, 0x04, 0x02, 0x04, ...}

[PcdsDynamicDefault.common.BoardY]
    gBoardModuleTokenSpaceGuid.PcdGpioPin|0x4
    gBoardModuleTokenSpaceGuid.PcdGpioInitValue|{0x00, 0x02, 0x01, 0x02, ...}

=============================
```

The SKU PCD is actually a dynamic PCD. The current implementation just puts all configuration data together without any size optimization, so even a one byte difference will cause a full SKU configuration to be duplicated. We can enhance the SkuIds PCD implementation. If there is any size concern in SkuIds PCD, the alternative could be: define one PcdsDynamicDefault, and let each platform update its own configuration there.

If a user finds it is hard to write the PCD initialization in DSC file, the alternative is to define a DynamicDefaultPCD with all zero as the initialized value, and then let each platform update the value in BoardInit.c. If this solution is chosen, there is no need to define this configuration to be a SKU-PCD.

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/KabylakeOpenBoardPkg/KabylakeRvp3/Library/BoardInitLib/PeiKabylakeRvp3InitPreMemLib.c)

```
====== PeiKabylakeRvp3InitPreMemLib.c ======
EFI_STATUS
EFIAPI
KabylakeRvp3InitPreMem (
  VOID
  )
{
  PcdSet32S (PcdPcie0WakeGpioNo, 0);
  PcdSet8S  (PcdPcie0HoldRstExpanderNo, 0);
  PcdSet32S (PcdPcie0HoldRstGpioNo, 8);

  PcdSet32S (PcdMrcRcompResistor, (UINTN) RcompResistorSklRvp1);
  PcdSet32S (PcdMrcRcompTarget, (UINTN) RcompTargetSklRvp1);

  ......
  PcdSet32S (PcdMrcSpdData, (UINTN) mSkylakeRvp3Spd110);
  PcdSet16S (PcdMrcSpdDataSize, mSkylakeRvp3Spd110Size);

  PcdSetBoolS (PcdIoExpanderPresent, TRUE);
```

```
  return EFI_SUCCESS;
}
====== PeiKabylakeRvp3InitPreMemLib.c ======
```

Board Detection and Initialization Flow Summary

1) The board detection in PlatformInit calls all BoardDetect() APIs registered by each board code. But only one of them will detect the board at runtime. If the board is detected the SkuId PCD is set to indicate the current configuration. After that the SkuIds PCD takes effect.
2) The detected board code registers a set of BoardInit. As such when PlatformInit calls BoardInit function, the registered board specific BoardInit function will be called.
3) The detected board code also sets a set of dynamic PCDs. These PCDs can also be used by common board code.

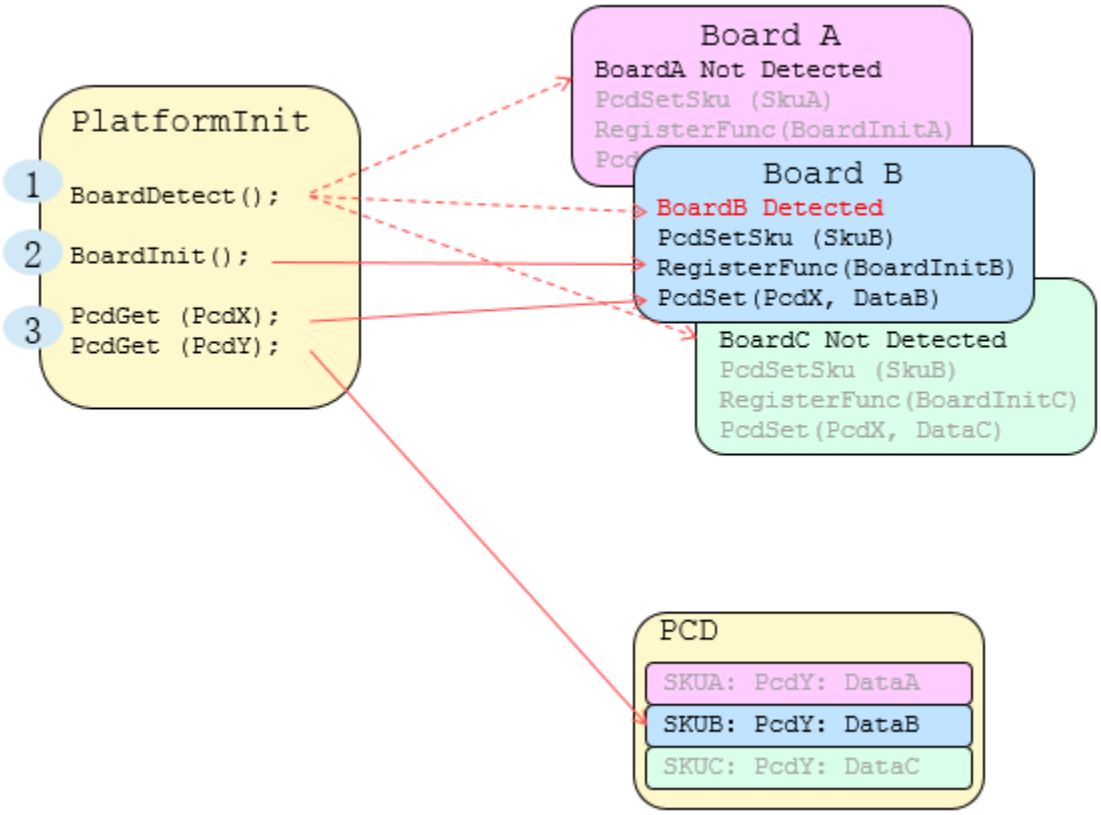

Figure 11 Board detection and initialization flow summary

Summary

This section introduces porting – board specific initialization.

Security

Once the board porting work is finished, the minimal security check work should be done to make sure there is not security hole. We described the basic security design guide in [SecurityDesign]. Some other security best practice can be found in [SecureSmmComm] [MemoryProtection].

Chipsec

[CHIPSEC] is a framework for security assessment of hardware and firmware components on the platform, enabling security research, testing, and forensics. Intel originally created it to help internal teams find and fix vulnerabilities in platform hardware and firmware.

A platform BIOS should run CHIPSEC before release.

HSTI

Microsoft Hardware Security Test Interface [HSTI] provides a set interfaces required by the Microsoft Windows Hardware Certification Requirement.

EDKII provides a generic HSTI definition at https://github.com/tianocore/edk2/blob/master/MdePkg/Include/IndustryStandard/Hsti.h and a DXE phase library to help construct the HSTI table at (https://github.com/tianocore/edk2/tree/master/MdePkg/Library/DxeHstiLib)

A silicon HSTI driver, such as https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Silicon/Intel/KabylakeSiliconPkg/Hsti/Dxe, should produce an HSTI table as `PLATFORM_SECURITY_ROLE_PLATFORM_REFERENCE`. A platform HSTI driver, such as https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Hsti/HstiIbvPlatformDxe, should produce an HSTI table as `PLATFORM_SECURITY_ROLE_PLATFORM_IBV`.

A platform BIOS should report HSTI and make sure no errors are reported in a HSTI table before release.

WSMT

In order to mitigate SMM communication buffer security issues on Microsoft Virtualization Based Security (VBS) in Windows 10, a platform BIOS needs to check all SMI handlers to ensure that all SMI handlers use a fixed communication buffer and report this fact via the Windows Security Mitigations Table (WSMT). [WSMT]

The latest EDKII core follows the WSMT recommendations and enables a fixed communication buffer. [SecureSmmComm]

A platform BIOS must check all SMI handlers and report the WSMT table.

Memory Attribute Table

In [MemoryMap], we discussed how to report the runtime memory attribute by using EFI_MEMORY_ATTRIBUTES_TABLE so that an OS can apply the protection for the runtime code and data pages for UEFI runtime content.

A platform BIOS should set 4KiB alignment for a runtime image and report the details in EFI_MEMORY_ATTRIBUTES_TABLE.

SMM Memory Attribute Table

In [MemoryProtection][SMMProtection], we discussed how to report the SMM memory attribute by using EDKII_PI_MEMORY_ATTRIBUTES_TABLE so that the SMM CPU driver can set up the page tables to protect SMM code and data.

A platform BIOS should set 4KiB alignment for each SMM image, set static SMM paging, and report this information via EDKII_PI_MEMORY_ATTRIBUTES_TABLE.

3rd Party Drivers

In order to maintain the platform authentication state, any 3rd party option ROM (OROM) must NOT be dispatch before EndOfDxe. The 3rd party option ROM includes a PCI OROM on the 3rd party PCI card. If a PCI OROM is integrated inside of the BIOS, it is NOT considered as a 3rd party option ROM and it MAY be dispatched before EndOfDxe. If there is a deferred OROM, the platform BDS may use EfiBootManagerDispatchDeferredImages() to dispatch the driver in this OROM after EndOfDxe. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Bds/Library/DxePlatformBootManagerLib/BdsPlatform.c)

Trusted Console

A platform should define a set of trusted consoles. The trusted console should be connected by BDS before EndOfDxe based upon the platform requirements. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Bds/Library/DxePlatformBootManagerLib/BdsPlatform.c) A trusted console means: 1) Integrated device such as a PS2/USB keyboard/mouse without any option ROM, 2) A chipset-integrated video card which is soldered to the system board and whose driver is inside of the BIOS instead of a separate PCI Option ROM flash container, 3) A 3rd party video card which is soldered to the system board and whose driver is inside of the BIOS instead of in a PCI Option ROM flash container. The trusted console may support features, such as TCG PP and OPAL Password. If a remote console is used as a trusted console, additional authentication may be used, such as request of an administrator password.

Trusted Storage Device

A platform should define a set of trusted storage devices. The trusted storage device should be

connected by BDS before EndOfDxe based upon the platform need. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Bds/Library/DxePlatformBootManagerLib/BdsPlatform.c) A trusted storage means: 1) Integrated device such as a USB/ATA/NVMe/UFS/eMMC/SD device without any option ROM, 2) A chipset-integrated storage card (such as RAID) which is soldered to the system board and whose driver is inside of the BIOS instead of in a PCI Option ROM, 3) A 3rd party storage card (such as SCSI, FC) which is soldered to the system board and whose driver is inside of the BIOS instead of in a PCI Option ROM. A platform must support the capability to allow user input password, such as OPAL password, in order to unlock the storage device if the storage is locked. [OPAL]

DMA

In [VTd EDKII], we discussed how to configure a DMA remapping table, such as an I/O memory management unit (IOMMU), for EDKII to prevent DMA-based attacks.

A platform must disable all DMA (PCI BME) before EndOfDxe. If possible, a platform should configure VTd to prevent illegal DMA access. If VTd is not set up, the platform should disable the DMA on the device if it is not a boot device (e.g., disconnect device).

Variable Usage

In [AUTH VARIABLE], we discussed the robust considerations for the variable management.

A platform must evaluate all the variables. The RT and NV attributes must be set only if it is necessary. A platform may use EDKII_VARIABLE_LOCK_PROTOCOL and/or EDKII_SMM_VAR_CHECK_PROTOCOL to protect the variable. A platform may define a reasonable variable quota to prevent variable out-of-resource conditions.

SMI Handler

In [SMMProtection], we discussed how to enable SMI handler profiling to detect how many SMI handlers are available in a given boot.

A platform BIOS should evaluate all the SMI handlers to see if there is a need to expose them, and remove all the unnecessary SMI handlers.

TPM MOR

In [TPM2 EDKII], we discussed how to support TPM2 in EDKII BIOS.

A platform must support cleaning memory just after memory initialization. A platform must support a secure memory-only reset (MOR) requirement which is documented at [SecureMOR]. Besides memory cleaning, a platform must send TPer Reset to a storage device if there is a MOR request. [TCG SIIS]

Firmware Update

In [FirmwareUpdate], we discussed how to support secure firmware update.

If a platform BIOS needs to support firmware update, we recommend it uses signed firmware update.

Other

In [SecurityEnhancement] and [MemoryProtection], we also discussed some other security technologies, such as StackCheck, HeapGuard, NonExecutable memory, Address Space Randomization. These technologies may be used to help mitigate other security vulnerabilities, too.

Summary

This section introduces assessing the security of a platform.

Tuning and Profiling

Flash Image Size

After the EDKII build is finished, the build tool generates the FV space information.

```
====== FV Space Information ======
FSP_S [57%Full] 327680 total, 188528 used, 139152 free
FVMAIN [98%Full] 2621440 total, 2594232 used, 27208 free
MICROCODE_FV [26%Full] 720896 total, 192608 used, 528288 free
FVMAIN2_COMPACT [%Full] 589824 total, 296 used, 589528 free
FVRECOVERY [65%Full] 131072 total, 85472 used, 45600 free
FVMAIN_COMPACT [26%Full] 2555904 total, 670512 used, 1885392 free
FVRECOVERY3_COMPACT [3%Full] 1507328 total, 48400 used, 1458928 free
FVRECOVERY2 [16%Full] 1179648 total, 196856 used, 982792 free
FVRECOVERY3 [66%Full] 196608 total, 131232 used, 65376 free
FVMAIN2 [%Full] 65536 total, 120 used, 65416 free
====== FV Space Information ======
```

If a developer has interest in the content in a FV (such as FVRECOVERY2), he can use the base tool – VolInfo (https://github.com/tianocore/edk2/tree/master/BaseTools/Source/C/VolInfo) to dump the detailed Firmware Volume information, including the FV header, each file and the respective sections.

```
====== VolInfo FVRECOVERY2.FV ======
VolInfo Version 1.0 Build Build 24507
Signature:         _FVH (4856465F)
Attributes:        7FEFF
       EFI_FVB2_READ_DISABLED_CAP
......
Header Length:          0x00000048
File System ID:         8c8ce578-8a3d-4f1c-9935-896185c32dd3
Revision:               0x0002
Number of Blocks:       0x00000012
Block Length:           0x00010000
Total Volume Size:      0x00120000
============================================================
......
============================================================
File Name:         9FAAD0FF-0E0C-4885-A738-BAB4E4FA1E66
File Offset:       0x0001FCE8
File Length:       0x0001040A
File Attributes:   0x10
File State:        0xF8
        EFI_FILE_DATA_VALID
File Type:         0x06  EFI_FV_FILETYPE_PEIM
------------------------------------------------------------
  Type:  EFI_SECTION_PEI_DEPEX
  Size:  0x00000028
        PUSH
        7408D748-FC8C-4EE6-9288-C4BEC092A410
        PUSH
        01F34D25-4DE2-23AD-3FF3-36353FF323F1
        AND
        END DEPEX
------------------------------------------------------------
  Type:  EFI_SECTION_RAW
  Size:  0x00000014
```

```
------------------------------------------------------------
  Type:  EFI_SECTION_PE32
  Size:  0x00010384
------------------------------------------------------------
  Type:  EFI_SECTION_USER_INTERFACE
  Size:  0x00000024
  String: FspmWrapperPeim
------------------------------------------------------------
  Type:  EFI_SECTION_VERSION
  Size:  0x0000000E
  Build Number:  0x00
  Version Strg:  1
There are a total of 19 files in this FV
====== VolInfo FVRECOVERY2.FV ======
```

For each EFI file, the EDKII build generates a MAP file. If a developer has interest for a specific EFI file (such as FspwWrapperPeim), the map can provide the final information of the image.

```
====== FspmWrapperPeim.map ======
......
 Start         Length     Name                   Class
 0001:00000000 00000185H .text                   CODE
 0001:00000190 0000837aH .text$mn                CODE
 0002:00000000 00006a64H .rdata                  DATA
 0002:00006a64 00000138H .rdata$zzzdbg           DATA
 0003:00000000 000007c8H .data                   DATA
 0003:000007c8 00000012H .bss                    DATA

  Address         Publics by Value              Rva+Base       Lib:Object

0001:00000000       _InternalMemCopyMem        00000260
BaseMemoryLibRepStr:CopyMem.obj
 0001:00000040       _InternalMemZeroMem        000002a0
......
0001:00001d42       _InternalPrintVariableData 00001fa2 f
PeiFspPolicyUpdateLib:PeiFspPolicyUpdateLib.obj
 0001:00001d8b       _InstallPlatformHsioPtssTable 00001feb f
PeiFspPolicyUpdateLib:PeiPchPolicyUpdatePreMem.obj
 0001:000023ee       _PeiFspMiscUpdUpdatePreMem 0000264e f
PeiFspPolicyUpdateLib:PeiFspMiscUpdUpdateLib.obj
 0001:00002489       _PeiFspSaPolicyUpdatePreMem 000026e9 f
PeiFspPolicyUpdateLib:PeiSaPolicyUpdatePreMem.obj
......
0002:00006518       _mRcompResistorSklRvp1     0000ec98
PeiSaPolicyLib:PeiSaPolicyLibSample.obj
 0002:00006520       _mDqsMapCpu2DramSklRvp     0000eca0
PeiSaPolicyLib:PeiSaPolicyLibSample.obj
 0002:00006530       _mDqByteMapSkl             0000ecb0
PeiSaPolicyLib:PeiSaPolicyLibSample.obj
 0002:00006548       _mRcompTargetSklRvp1       0000ecc8
PeiSaPolicyLib:PeiSaPolicyLibSample.obj
 0002:00006558       _mSkylakeRvp3Spd           0000ecd8
PeiSaPolicyLib:PeiSaPolicyLibSample.obj
......
====== FspmWrapperPeim.map ======
```

Now it is time to do the analysis. If there is a global data item that should not be included in the build, we can check if `GLOBAL_REMOVE_IF_UNREFERENCED` is added. If there is a very big function, we can see if we can use some optimization to make it smaller.

Memory Consumption

In [EdkIIProfile], we discussed how to calculate the memory consumption in a UEFI or SMM environment. We recommend a platform BIOS include the profile tool to check if the memory consumption is as expected.

Memory Leak

In [EdkIIProfile], we discussed how to detect memory leaks in a UEFI or SMM environment. We recommend a platform BIOS include the profile tool in order to check if there is a memory leak in the BIOS.

Boot Performance

In [EdkIIProfile], we discussed how to get boot performance data in a EDKII BIOS. We recommend a platform BIOS include the profile tool to check if the boot performance is expected.

Summary

This section introduces assessing the size and performance of a platform.

Testability

Test point in boot flow

In system boot, there are some important milestones. We define a set of test points to check the system state at these various locations. (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/TestPointCheckLib.h)

Table 2 Platform Test Point

Phase	Function	Check item
SEC	TestPointTempMemoryInitDone ()	TempRam Information
PEI	TestPointDebugInitDone ()	Debug Output
	TestPointMemoryDiscovered ()	Memory Resource Information MTRR Setting (for PEI) SMRAM Hob Infomration FV information.
	TestPointEndOfPei ()	MTRR Setting (for DXE) System Resource Information
DXE	TestPointPciEnumerationDone ()	PCI Resource Information
	TestPointEndOfDxe ()	UEFI Image Information No 3rd Party PCI OROM Trusted Console Variable Device Path List
	TestPointDxeSmmReadyToLock ()	
	TestPointReadyToBoot ()	UEFI Memory Map GCD Map Memory Type Information UEFI Console Variable UEFI Boot Variable System Device Information UEFI Memory Attribute Table ACPI Table DMAR Table WSMT Table HSTI information. ESRT information.
	TestPointExitBootServices ()	
SMM	TestPointSmmEndOfDxe ()	SMM Image Information
	TestPointSmmReadyToLock ()	SMRAM Information SMRR Setting UEFI Memory Map for SMM GCD Map for SMM SMM Memory Attribute Table

	TestPointSmmReadyToBoot ()	
	TestPointSmmExitBootServices ()	

Because the PlatformInitPei, PlatformInitDxe and PlatformInitSmm control the platform initialization flow, we link the test point check library into those modules. (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Test/Library/TestPointCheckLib)

We not only dump these important test information, but we also report the test point check results via ADAPTER_INFO_PLATFORM_TEST_POINT (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Library/TestPointCheckLib.h). This is a HSTI-like reporting machanism, so that we can use a tool – TestPointDumpApp (https://github.com/tianocore/edk2-platforms/tree/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Test/TestPointDumpApp) to dump the test result in UEFI SHELL environment.

The purpose of these test point checks is to help a platform developer catch any mistake or missing features before a BIOS release.

Summary
This section talks about the testability of the open platform.

Core Module Selection

Mandatory V.S. Optional

EDKII source code has lots of modules (driver and library). Not all of them are needed in the final BIOS image.

We made an analysis on current EDKII core package usage in the EDKII Kabylake platform. In this analysis, EDKII "Core package" means `CryptoPkg, FatBinPkg, FatPkg, IntelFrameworkModulePkg, IntelFrameworkPkg, IntelFsp2Pkg, IntelFsp2WrapperPkg, IntelFspPkg, IntelFspWrapperPkg, IntelSiliconPkg, MdeModulePkg, MdePkg, NetworkPkg, PcAtChipsetPkg, PerformancePkg, SecurityPkg, ShellBinPkg, ShellPkg, SignedCapsulePkg, SourceLevelDebugPkg, UefiCpuPkg.`

Table 3 Core Module usage

	Kabylake (ShellOnly)	Kabylake (OS Boot)	All
Module Count	57	68	259
Library Count	83	95	309
Include File Count	368	422	838
Module File Count	492	641	2426
Library File Count	1131	1204	2693
All File Count	2045	2321	6098

For the final result, only ~60 core module are used in the Kabylake MinPlatform. About 200 modules are not needed. These are optional.

This data can be obtained from a tool - CheckCodeBase.py (https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Tools/Help/CheckCodeBase.py). It helps analyze how many files/modules are used in a tree.

We give a sample list of x86 min BIOS PEI/DXE module as Appendix B.

Deprecated module

Some EDKII core modules are marked as deprecated. If so, please try to not use them.
For example:

IntelFrameworkModulePkg\Universal\BdsDxe
SecurityPkg\Tcg\MemoryOverwriteRequestControlLock
SecurityPkg\Tcg\TrEEConfig
SecurityPkg\Tcg\TrEEDxe
SecurityPkg\Tcg\TrEEPei
SecurityPkg\Tcg\TrEESmm

Deprecated packages:

EdkCompatibilityPkg
EdkShellBinPkg
EdkShellPkg

Packages will be deprecated:

IntelFrameworkPkg
IntelFrameworkModulePkg

Deprecated API

Because of the Security Development Lifecycle (SDL) requirement, we identified a set of banned functions in EDKII. The similar work in OS can be found at [Banned Function].

In EDKII, these banned functions are marked as deprecated API. They are decorated by the MACRO `DISABLE_NEW_DEPRECATED_INTERFACES`. We suggest a platform enables this MACRO to ensure that no deprecated APIs are used.

Table 4 EDKII Deprecated APIs

Category	Deprecated API	Replacement
BaseLib	StrCpy	StrCpyS
	StrnCpy	StrnCpys
	StrCat	StrCatS
	StrnCat	StrnCats
	AsciiStrCpy	AsciiStrCpyS
	AsciiStrnCpy	AsciiStrnCpys
	AsciiStrCat	AsciiStrCatS
	AsciiStrnCat	AsciiStrnCats
	UnicodeStrToAsciiStr	UnicodeStrToAsciiStrS UnicodeStrnToAsciiStrS
	AsciiStrToUnicodeStr	AsciiStrToUnicodeStrS AsciiStrnToUnicodeStrS
PcdLib	[Lib]PcdSet[Ex]8	[Lib]PcdSet[Ex]8S
	[Lib]PcdSet[Ex]16	[Lib]PcdSet[Ex]16S
	[Lib]PcdSet[Ex]32	[Lib]PcdSet[Ex]32S
	[Lib]PcdSet[Ex]64	[Lib]PcdSet[Ex]64S
	[Lib]PcdSet[Ex]Ptr	[Lib]PcdSet[Ex]PtrS
	[Lib]PcdSet[Ex]Bool	[Lib]PcdSet[Ex]BoolS
PrintLib	UnicodeValueToString	UnicodeValueToStringS
	AsciiValueToString	AsciiValueToStringS
UefiLib	GetVariable	GetVariable2
	GetEfiGlobalVariable	GetEfiGlobalVariable2

Core Module Override

Some core modules might not meet a particular platform requirement. A platform may create an Override directory to create a duplicate set of core modules. This is only acceptable as a short term solution when a core module has design issues and the core module owner cannot resolve the issue quickly.

In general, we do not recommend a platform owner overriding the core module directly and putting them into a platform directory because it might bring core synchronization/update issues later. We recommend a platform owner reporting issue to a core module owner (see maintainer list on tianocore.org) to address the problem and work with core module owner to design a new solution. This not only benefits the platform owner by removing the extra overhead to maintain a new driver, but it also benefits the core module owner by enhancing the core solution to be applicable for more platforms.

Summary

This section talks about the distinction between core and platform modules.

Summary

A summary of the recommended guidelines is shown below:

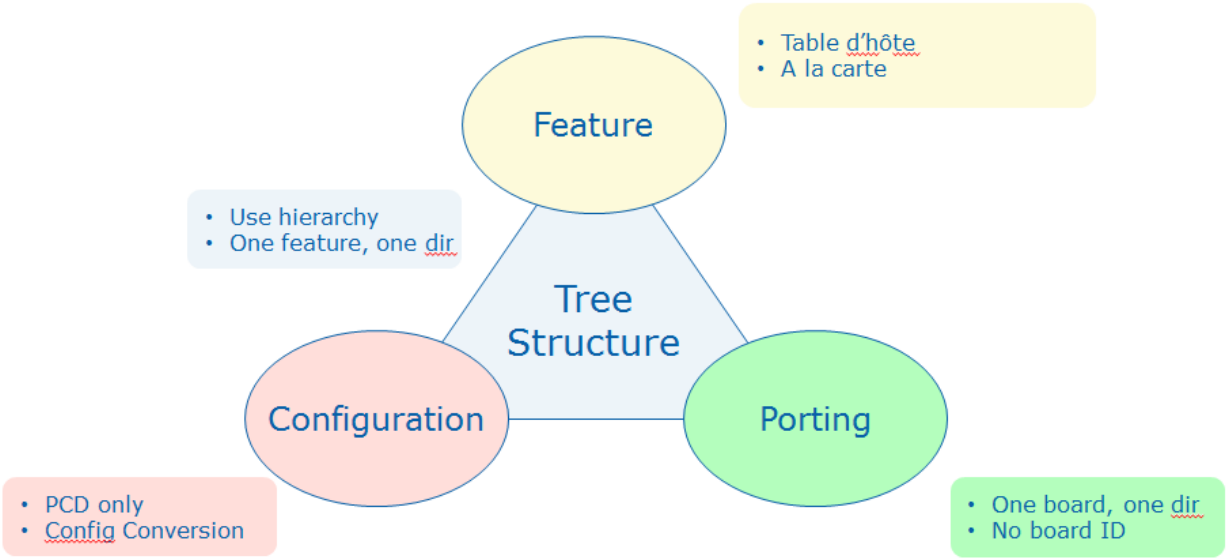

Figure 12 guideline summary

Appendix A – Open Platform Design Guideline

Platform Feature [F]

- [F1] Put one feature to one directory.
- [F2] Put advanced features to Features directory.
- [F3] Provide configuration options.

Policy Configuration [C]

- [C1] Don’t call GetVariable/SetVariable to get/set policy data.
- [C2] Use PCD to pass policy data in platform code.
- [C3] Let silicon provide default configuration and platform update it.
- [C4] Use silicon interface (Hob/PPI/Protocol/FSP UPD) for silicon only.
- [C5] Expose one configuration source to user/developer.

Board Specific Code [B]

- [B1] Don’t use a board specific check (BoardId/PlatformType) in any C file, except board specific driver.
- [B2] Don’t use a board specific check (BoardId/PlatformType) in any ASL, except board specific driver.
- [B3] Don’t use a board specific check (BoardId/PlatformType) in any VFR, except board specific driver.
- [B4] Put board specific configuration to BoradInit folder.
- [B5] Put board specific initialization to BoradInit folder.
- [B6] Put board specific ACPI table to BoradAcpiTable folder.
- [B7] Override board specific ACPI configuration in BoardAcpi folder.
- [B8] Override board specific VFR default value in BoardInit folder.
- [B9] Platform code should not have board specific knowledge and not depend upon board code.
- [B10] Create a Board folder and put all the board specific code there.

Secure By Default [S]

- [S1] Enable all silicon/platform security features by default. Especially, SMM Lock, Flash Lock, Chipset register lock, etc.
- [S2] Run CHIPSEC before release.
- [S3] Enable HSTI for Windows.
- [S4] Report WSMT for Windows Hypervisor.
- [S5] Report UEFI_MEMORY_ATTRIBUTE_TABLE for Windows Hypervisor.
- [S6] Report SMM_MEMORY_ATTRIBUTE_TABLE for SMM.

- [S7] Do not dispatch 3rd party driver before EndOfDxe.
- [S8] Define trusted console and trusted storage device based upon the need.
- [S9] Evaluate UEFI variable usage.
- [S10] Evaluate SMI handler.
- [S11] Enable TestPointCheckLib and check result.

Core module selection [M]

- [M1] Do not override EDKII core module.
- [M2] Do not use API deprecated by `DISABLE_NEW_DEPRECATED_INTERFACES`
- [M3] Do not use deprecated module in EDKII.

Appendix B – x86 Min BIOS Template

Many people asked how to create a minimal UEFI BIOS. Here is template for BASIC OS BOOT:

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Fdf/CorePeiBfvInclude.fdf)

```
====== CorePeiBfvInclude.dsc ======
INF  UefiCpuPkg/SecCore/SecCore.inf
INF  MdeModulePkg/Core/Pei/PeiMain.inf
====== CorePeiBfvInclude.dsc ======
```

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Fdf/CorePeiPreMemInclude.fdf)

```
====== CorePeiPreMemInclude.fdf ======
INF  MdeModulePkg/Universal/PCD/Pei/Pcd.inf
INF  MdeModulePkg/Universal/ReportStatusCodeRouter/Pei/ReportStatusCodeRouterPei.inf
INF  MdeModulePkg/Universal/StatusCodeHandler/Pei/StatusCodeHandlerPei.inf

INF  UefiCpuPkg/CpuIoPei/CpuIoPei.inf

INF  MdeModulePkg/Universal/PcatSingleSegmentPciCfg2Pei/PcatSingleSegmentPciCfg2Pei.inf
INF  MdeModulePkg/Universal/Variable/Pei/VariablePei.inf
INF  MdeModulePkg/Universal/FaultTolerantWritePei/FaultTolerantWritePei.inf

INF  MdeModulePkg/Core/DxeIplPeim/DxeIpl.inf

!if gPlatformModuleTokenSpaceGuid.PcdTpm2Enable == TRUE
INF  SecurityPkg/Tcg/Tcg2Config/Tcg2ConfigPei.inf
INF  SecurityPkg/Tcg/Tcg2Pei/Tcg2Pei.inf
!endif
====== CorePeiPreMemInclude.fdf ======
```

(https://github.com/tianocore/edk2-platforms/blob/devel-MinPlatform/Platform/Intel/MinPlatformPkg/Include/Fdf/CoreDxeInclude.fdf)

```
====== CoreDxeInclude.fdf ======
INF  MdeModulePkg/Core/Dxe/DxeMain.inf
INF  MdeModulePkg/Universal/PCD/Dxe/Pcd.inf

INF  UefiCpuPkg/CpuIo2Dxe/CpuIo2Dxe.inf
INF  PcAtChipsetPkg/8259InterruptControllerDxe/8259.inf
INF  MdeModulePkg/Universal/Metronome/Metronome.inf
INF  MdeModulePkg/Universal/WatchdogTimerDxe/WatchdogTimer.inf
INF  PcAtChipsetPkg/PcatRealTimeClockRuntimeDxe/PcatRealTimeClockRuntimeDxe.inf
INF  MdeModulePkg/Core/RuntimeDxe/RuntimeDxe.inf

!if gPlatformModuleTokenSpaceGuid.PcdBootToShellOnly == FALSE
INF  MdeModulePkg/Universal/FaultTolerantWriteDxe/FaultTolerantWriteSmm.inf
INF  MdeModulePkg/Universal/Variable/RuntimeDxe/VariableSmmRuntimeDxe.inf
INF  MdeModulePkg/Universal/Variable/RuntimeDxe/VariableSmm.inf
!else
INF  MdeModulePkg/Universal/Variable/EmuRuntimeDxe/EmuVariableRuntimeDxe.inf
!endif

INF  MdeModulePkg/Universal/MonotonicCounterRuntimeDxe/MonotonicCounterRuntimeDxe.inf
```

```
INF  MdeModulePkg/Universal/BdsDxe/BdsDxe.inf
INF  MdeModulePkg/Universal/DriverHealthManagerDxe/DriverHealthManagerDxe.inf
INF  MdeModulePkg/Universal/SecurityStubDxe/SecurityStubDxe.inf
INF  MdeModulePkg/Universal/CapsuleRuntimeDxe/CapsuleRuntimeDxe.inf

INF  UefiCpuPkg/CpuDxe/CpuDxe.inf

INF  MdeModulePkg/Universal/ResetSystemRuntimeDxe/ResetSystemRuntimeDxe.inf
INF  PcAtChipsetPkg/HpetTimerDxe/HpetTimerDxe.inf

INF  MdeModulePkg/Bus/Pci/PciHostBridgeDxe/PciHostBridgeDxe.inf
INF  MdeModulePkg/Bus/Pci/PciBusDxe/PciBusDxe.inf

INF  MdeModulePkg/Bus/Pci/SataControllerDxe/SataControllerDxe.inf
INF  MdeModulePkg/Bus/Ata/AtaBusDxe/AtaBusDxe.inf
INF  MdeModulePkg/Bus/Ata/AtaAtapiPassThru/AtaAtapiPassThru.inf

INF  MdeModulePkg/Bus/Pci/XhciDxe/XhciDxe.inf
INF  MdeModulePkg/Bus/Pci/EhciDxe/EhciDxe.inf
INF  MdeModulePkg/Bus/Pci/UhciDxe/UhciDxe.inf
INF  MdeModulePkg/Bus/Usb/UsbBusDxe/UsbBusDxe.inf
INF  MdeModulePkg/Bus/Usb/UsbMassStorageDxe/UsbMassStorageDxe.inf
INF  MdeModulePkg/Bus/Usb/UsbKbDxe/UsbKbDxe.inf

INF  MdeModulePkg/Universal/Disk/DiskIoDxe/DiskIoDxe.inf
INF  MdeModulePkg/Universal/Disk/PartitionDxe/PartitionDxe.inf
INF  MdeModulePkg/Universal/Disk/UnicodeCollation/EnglishDxe/EnglishDxe.inf
INF  FatBinPkg/EnhancedFatDxe/Fat.inf

INF  MdeModulePkg/Universal/Console/GraphicsOutputDxe/GraphicsOutputDxe.inf
INF  MdeModulePkg/Universal/Console/GraphicsConsoleDxe/GraphicsConsoleDxe.inf

INF  MdeModulePkg/Universal/Console/ConPlatformDxe/ConPlatformDxe.inf
INF  MdeModulePkg/Universal/Console/ConSplitterDxe/ConSplitterDxe.inf

INF  MdeModulePkg/Universal/DevicePathDxe/DevicePathDxe.inf

INF  MdeModulePkg/Universal/MemoryTest/NullMemoryTestDxe/NullMemoryTestDxe.inf

INF  MdeModulePkg/Universal/HiiDatabaseDxe/HiiDatabaseDxe.inf
INF  MdeModulePkg/Universal/SetupBrowserDxe/SetupBrowserDxe.inf
INF  MdeModulePkg/Universal/DisplayEngineDxe/DisplayEngineDxe.inf

INF  RuleOverride = UI MdeModulePkg/Application/UiApp/UiApp.inf
INF  MdeModulePkg/Application/BootManagerMenuApp/BootManagerMenuApp.inf

!if gPlatformModuleTokenSpaceGuid.PcdBootToShellOnly == FALSE
INF  MdeModulePkg/Core/PiSmmCore/PiSmmIpl.inf
INF  MdeModulePkg/Core/PiSmmCore/PiSmmCore.inf
INF  MdeModulePkg/Universal/ReportStatusCodeRouter/Smm/ReportStatusCodeRouterSmm.inf
INF  MdeModulePkg/Universal/StatusCodeHandler/Smm/StatusCodeHandlerSmm.inf
INF  UefiCpuPkg/PiSmmCpuDxeSmm/PiSmmCpuDxeSmm.inf
INF  UefiCpuPkg/CpuIo2Smm/CpuIo2Smm.inf
INF  MdeModulePkg/Universal/SmmCommunicationBufferDxe/SmmCommunicationBufferDxe.inf

INF  MdeModulePkg/Universal/Acpi/AcpiTableDxe/AcpiTableDxe.inf
INF
MdeModulePkg/Universal/Acpi/FirmwarePerformanceDataTableDxe/FirmwarePerformanceDxe.inf
```

```
INF
MdeModulePkg/Universal/Acpi/FirmwarePerformanceDataTableSmm/FirmwarePerformanceSmm.inf
INF
MdeModulePkg/Universal/Acpi/BootGraphicsResourceTableDxe/BootGraphicsResourceTableDxe.inf
!endif

!if gPlatformModuleTokenSpaceGuid.PcdUefiSecureBootEnable == TRUE
INF  SecurityPkg/VariableAuthenticated/SecureBootConfigDxe/SecureBootConfigDxe.inf
!endif

!if gPlatformModuleTokenSpaceGuid.PcdTpm2Enable == TRUE
INF  SecurityPkg/Tcg/MemoryOverwriteControl/TcgMor.inf
INF  SecurityPkg/Tcg/Tcg2Dxe/Tcg2Dxe.inf
INF  SecurityPkg/Tcg/Tcg2Smm/Tcg2Smm.inf
INF  SecurityPkg/Tcg/Tcg2Config/Tcg2ConfigDxe.inf
!endif
====== CoreDxeInclude.fdf ======
```

Conclusion

Previous IA firmware examples are sometimes complex and inconsistent. We are trying to provide some general guidelines for an open source IA firmware design in order to create a simple and consistent platform code set and to scale the solution to all Intel platforms. These platforms include Intel ATOM, Intel Core-i7, and Intel XEON generation.

Acknowledgement

Many people are involved in the platform architecture design. We would like to thank Isaac W Oram (Intel Data Center Group, DCG), Brett Wang (Intel DCG), Daocheng Bu (Intel DCG), Thad Gillispie (Intel DCG), and Maurice Ma (Intel Internet of Thing Group, IOTG) who were involved in the design discussions and gave us valuable feedback.

Glossary

CMOS - Complementary Metal Oxide Semiconductor. A storage for legacy BIOS.

DMA – Direct Memory Access.

DMAR – DMA Remapping Reporting Table. See [Intel VT-d]

HPET – High Precission Event Timer. See [ACPI].

HSTI – Hardware Security Test Interface. See [HSTI]

IPL – Initial program loader.

MCFG – Memory Mapped Configuration Space Access Table. See [ACPI].

MOR – (TPM) Memory Overwrite Request. See [MOR] and [SecureMOR]

OPAL - Storage Work Group Storage Security Subsystem Class: Opal. See [OPAL]

OROM – Option ROM.

PCD – Platform Configuration Database. See [UEFI PI Specification].

PI – Platform Initialization. Volume 1-5 of the UEFI PI specifications.

TPer – Trusted Peripheral. See [TCG SIIS]

TPM – Trusted Platform Module. See [TPM2]

UEFI – Unified Extensible Firmware Interface. Firmware interface between the platform and the operating system.

UPD – Updatable production data. See [FSP EAS].

VPD – Vital production data. See [FSP EAS].

WSMT – Windows SMM Mitigation Table. See [WSMT]

References

[ACPI] Advanced Configuration and Power Interface, vesion 6.0, www.uefi.org

[AUTH VARIABLE] Yao, Zimmer, Zeng, "A Tour Beyond BIOS Implementing UEFI Authenticated Variables in SMM with EDKII – Verion 2," September 2015 https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_UEFI_Authenticated_Variables_in_SMM_with_EDKII_V2.pdf

[Banned Function] Microsoft Security Development Lifecycle (SDL) Banned Function Calls, https://msdn.microsoft.com/en-us/library/bb288454.aspx

[Baytrail Data sheet] Intel® Atom™ Processor E3800: Datasheet, http://www.intel.com/content/www/us/en/embedded/products/bay-trail/atom-e3800-family-datasheet.html

[Braswell Data sheet] N-series Pentium® and Celeron® Processors Datasheet, http://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-datasheet-vol-1.html, http://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-datasheet-vol-2.html, http://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-datasheet-vol-3.html

[CHIPSEC] CHIPSEC: Platform Security Assessment Framework http://www.intelsecurity.com/advanced-threat-research/chipsec.html

[COREBOOT] coreboot firmware www.coreboot.org

[EDK2] UEFI Developer Kit www.tianocore.org

[EDKII specification] A set of specification describe EDKII DEC/INF/DSC/FDF file format, as well as EDKII BUILD. http://tianocore.sourceforge.net/wiki/EDK_II_Specifications

[EdkIIProfile] Yao, Zimmer, Zeng, Fan, A Tour Beyond BIOS Implementing Profiling in EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_Profiling_in_EDK_II.pdf

[EMBED] Sun, Jones, Reinauer, Zimmer, "Embedded Firmware Solutions: Development Best Practices for the Internet of Things," Apress, January 2015, ISBN 978-1-4842-0071-1

[FirmwareUpdate] Yao, Zimmer, A Tour Beyond BIOS- Capsule Update and Recovery in EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Capsule_Update_and_Recovery_in_EDK_II.pdf

[FSP] Intel Firmware Support Package http://www.intel.com/content/www/us/en/intelligent-systems/intel-firmware-support-package/intel-fsp-overview.html

[FSP EAS] FSP External Architecture Specification
http://www.intel.com/content/www/us/en/embedded/software/fsp/fsp-architecture-spec-v1-1.html

[FSP Consumer] Yao, Zimmer, Rangarajan, Ma, Estrada, Mudusuru, "A_Tour_Beyond_BIOS_Using_the_Intel_Firmware_Support_Package_with_the_EFI_Developer_Kit_II_(FSP1.1)" http://firmware.intel.com

[HSTI] Hardware Security Testability Specification https://msdn.microsoft.com/en-us/library/windows/hardware/mt712332.aspx

[Intel Graphic OpRegion] Intel® Integrated Graphics Device - OpRegion Specification
https://01.org/linuxgraphics/documentation/intel%C2%AE-integrated-graphics-device-opregion-specification

[IA32 Manual] Intel® 64 and IA-32 Architectures Software Developer Manuals
http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[Intel DCMI] DCMI Host Interface Specification
http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/dcmi-hi-1-0-spec.pdf

[Intel SGX] Intel® Software Guard Extensions Programming Reference
https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[Intel TraceHub] Intel® Trace Hub Developer's Manual
https://software.intel.com/sites/default/files/managed/d3/3c/intel-th-developer-manual.pdf

[Intel TXT] Intel® Trusted Execution Technology (Intel® TXT) Software Development Guide
http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf

[Intel VT-d] Intel® Virtualization Technology for Directed I/O specification, Rev 2.3
http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/vt-directed-io-spec.html

[Kabylake SA data sheet] 7th Generation Intel® Processor Families
https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-desktop-s-processor-lines-datasheet-vol-1.html
https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-desktop-s-processor-lines-datasheet-vol-2.html

https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-mobile-u-y-processor-lines-datasheet-vol-1.html
https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-mobile-u-y-processor-lines-datasheet-vol-2.html

[Kabylake PCH data sheet] 7th Generation Intel® Processor Families I/O Platform
https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-mobile-u-y-processor-lines-i-o-datasheet-vol-1.html
https://www.intel.com/content/www/us/en/processors/core/7th-gen-core-family-mobile-u-y-processor-lines-i-o-datasheet-vol-2.html

[MemoryMap] Yao, Zimmer, A Tour Beyond BIOS Memory Map And Practices in UEFI BIOS, https://github.com/tianocoredocs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Memory_Map_And_Practices_in_UEFI_BIOS_V2.pdf

[MemoryProtection] Yao, Zimmer, A Tour Beyond BIOS- Memory Protection in UEFI BIOS, https://www.gitbook.com/book/edk2-docs/a-tour-beyond-bios-memory-protection-in-uefi-bios/details

[MOR] TCG Platform Reset Attack Mitigation Specification, https://www.trustedcomputinggroup.org/wp-content/uploads/Platform-Reset-Attack-Mitigation-Specification.pdf

[OSTS1] Vincent Zimmer, “Open Source IA Firmware Directions”, Open Source Technology Summit, 2015

[Quark data sheet] Intel® Quark™ SoC X1000: Datasheet, http://www.intel.com/content/www/us/en/embedded/products/quark/quark-x1000-datasheet.html

[SECURE BOOT] Nystrom, Nicoles, Zimmer, “UEFI Networking and Pre-OS Security,” Intel Technology Journal, October 2011

[SecurityEnhancement] Yao, Zimmer, A Tour Beyond BIOS Securiy Enhancement to Mitigate Buffer Overflow in UEFI, https://github.com/tianocoredocs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Security_Enhancement_to_Mitigate_Buffer_Overflow_in_UEFI.pdf

[SecurityDesign] Yao, Zimmer, A Tour Beyond BIOS Security Design Guide in EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Security_Design_Guide_in_EDK_II.pdf

[SecureMOR] Secure MOR implementation, https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/device-guard-requirements

[SecureSmmComm] Yao, Zimmer, Zeng, A tour beyond BIOS secure SMM communciation, https://github.com/tianocore-

docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Secure_SMM_Communication.pdf

[Skylake SA data sheet] 7th Generation Intel® Processor Families
https://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-datasheet-vol-1.html?wapkw=6th+generation+intel+core+datasheet
https://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-datasheet-vol-2.html?wapkw=6th+generation+intel+core+datasheet

[Skylake PCH data sheet] 6th Generation Intel® Processor Families I/O Platform
https://www.intel.com/content/www/us/en/chipsets/100-series-chipset-datasheet-vol-1.html?wapkw=6th+generation+intel+core+datasheet
https://www.intel.com/content/www/us/en/chipsets/100-series-chipset-datasheet-vol-2.html?wapkw=6th+generation+intel+core+datasheet
https://www.intel.eu/content/www/eu/en/processors/core/6th-gen-core-pch-u-y-io-datasheet-vol-1.html
https://www.intel.eu/content/www/eu/en/processors/core/6th-gen-core-pch-u-y-io-datasheet-vol-2.html

[SMMProtection] Yao, SMM Protection in EDKII.
http://www.uefi.org/sites/default/files/resources/Jiewen%20Yao%20-%20SMM%20Protection%20in%20%20EDKII_Intel.pdf

[TCG OPAL] Storage Work Group Storage Security Subsystem Class: Opal, Version 2.01 Final, Revision 1.00, https://trustedcomputinggroup.org/wp-content/uploads/TCG_Storage-Opal_SSC_v2.01_rev1.00.pdf

[TCG SIIS] TCG Storage Interface Interactions Specification, Version 1.06, Revision 1.08, https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_SWG_SIIS_Version_1_06_Revision_1_08_public-review.pdf

[TPM2] Trusted Platform Module Library Specification, Family “2.0”, Level 00, Revision 01.38 – September 2016, https://trustedcomputinggroup.org/tpm-library-specification/

[TPM2 PFP] PC Client Specific Platform Firmware Profile Specification Family “2.0”, Level 00 Revision 1.03 Version 51, https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf

[TPM2 EDKII] Yao, Zimmer, A Tour Beyond BIOS with the UEFI TPM2 Support in EDKII
https://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Implementing_TPM2_Support_in_EDKII.pdf

[UEFI Plugfest1] Leif Lindholm, “A Common Platforms Tree”, UEFI Plugfest, 2015

[UEFI] Unified Extensible Firmware Interface (UEFI) Specification, Version 2.5
www.uefi.org

[UEFI Book] Zimmer, et al, “Beyond BIOS: Developing with the Unified Extensible Firmware Interface,” 2nd edition, Intel Press, January 2011

[UEFI PI Specification] UEFI Platform Initialization (PI) Specifications, volumes 1-5, Version 1.4 www.uefi.org

[VTd EDKII] Yao, Zimmer, A Tour Beyond BIOS Using Intel VT-d for DMA Protection, https://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Using_Intel_VT-d_for_DMA_Protection.pdf

[WHCK System] Windows Hardware Certification Requirements for Client and Server Systems http://msdn.microsoft.com/en-us/library/windows/hardware/jj128256.aspx

[WSMT] Windows SMM Security Table, https://msdn.microsoft.com/en-us/library/windows/hardware/dn495660(v=vs.85).aspx#wsmt http://download.microsoft.com/download/1/8/A/18A21244-EB67-4538-BAA2-1A54E0E490B6/WSMT.docx

Authors

Jiewen Yao (jiewen.yao@intel.com) is an EDKII BIOS architect, EDKII TPM2 module maintainer, ACPI/S3 module maintainer, and FSP package owner with Software and Services Group (SSG) at Intel Corporation.

Vincent J. Zimmer (vincent.zimmer@intel.com) is a Senior Principal Engineer with the Software and Services Group (SSG) at Intel Corporation. Vincent chairs the UEFI Security and Networking Subteams in the UEFI Forum.

Michael A Kubacki (michael.a.kubacki@intel.com) is a BIOS architect in the Client Components Group (CCG) at Intel Corporation.

Amy Chan (amy.chan@intel.com) is a BIOS architect in the Client Components Group (CCG) at Intel Corporation.

Rangasai V Chaganty (rangasai.v.chaganty@intel.com) is a BIOS architect in the Client Components Group (CCG) at Intel Corporation.

Chasel Chiu (chasel.chiu@intel.com) is a BIOS architect in the Client Components Group (CCG) at Intel Corporation.