**AptioV PEI&SEC Phase源代码分析**

**(参考PI CIS V1.5)**

目录

[1 PI Architecture firmware Phase 2](#_Toc486841732)

[2 SEC 2](#_Toc486841733)

[3 PEI Overview 2](#_Toc486841734)

[**3.1** **Introduction** 2](#_Toc486841735)

[3.2 PEI架构 3](#_Toc486841736)

[**3.3** **Goal** 3](#_Toc486841737)

[3.4 **PEI consists of**（见Figure1） 3](#_Toc486841738)

[3.5 **PEI Service Table**（**struct** \_EFI\_PEI\_SERVICES），**not rich** because of philosophy of PEI phase 3](#_Toc486841739)

[3.6 **PEI Foundation（被设计成硬件平台无关）的工作内容** 5](#_Toc486841740)

[**3.7** **PEI Dispacher（状态机）** 5](#_Toc486841741)

[**3.8** **PEIM** 6](#_Toc486841742)

[**3.9** **PPI（PEIM间的通信）** 6](#_Toc486841743)

[**3.10** **Firmware Volume -> Firmware File System -> Data Section** 6](#_Toc486841744)

[4 PEI Service Table（PEI Foundation创建；services are not rich；使用PEIM和PPI，须传递“IN VOID \*PeiServices”指针）。 6](#_Toc486841745)

[5 PEI Service 7](#_Toc486841746)

[5.1 PPI FUNCTIONS 7](#_Toc486841747)

[5.2 Boot Mode Functions 8](#_Toc486841748)

[5.3 HOB Functions 8](#_Toc486841749)

[5.4 Firmware Volume Functions 9](#_Toc486841750)

[5.5 PEI Memory Functions 10](#_Toc486841751)

[5.6 Status Code 11](#_Toc486841752)

[5.7 Reset 11](#_Toc486841753)

[5.8 I/O Abstractions 11](#_Toc486841754)

[5.9 Future Installed Services 11](#_Toc486841755)

[6 PEI Foundation（含Dispatcher） 11](#_Toc486841756)

[6.1 PEI Foundation entry point 11](#_Toc486841757)

[**6.2** **如何获取 \*\*PeiServices** 12](#_Toc486841758)

[**6.3** **Pei Dispatcher Introduction** 13](#_Toc486841759)

[**6.4** **PEIM Odering & depex grammar** 13](#_Toc486841760)

[**6.5** **Dispatcher Algorithm** 14](#_Toc486841761)

[7 PEIMs & PPI的安装与获取 16](#_Toc486841762)

[7.1 **PEI阶段**，PEIMs是execution unit，而PEI Foundation是execution unit调度器。 16](#_Toc486841763)

[7.2 **PEIMs间通过PPI通信**，还可使用全局的PeiServices提供的服务。 16](#_Toc486841764)

[7.3 **与DXE Driver类似，PEIM有5类**： 16](#_Toc486841765)

[**7.4** **PEIM Structure** 17](#_Toc486841766)

[**7.5** **PEIM Invocation Entry Point** 17](#_Toc486841767)

[7.6 PEIM descriptor（EFI\_PEI\_DESCRIPTOR） 18](#_Toc486841768)

[8 Architecture PPI 19](#_Toc486841769)

[8.1 一般来讲，Architecture PPI是提供给PEI Foundation用的 19](#_Toc486841770)

[8.2 PEI Foundation和PEI Dispatcher通过PPI来协调PEIM完成PEI阶段的工作，如下图 19](#_Toc486841771)

[9 Additional PPI 20](#_Toc486841772)

[10 PEI to DXE handoff 21](#_Toc486841773)

[10.1 Discover and dispatch the DXE Foundation 21](#_Toc486841774)

[10.2 HobList必须包含如下类型的HOB 21](#_Toc486841775)

[10.3 DXE IPL PPI对CPU状态的要求 22](#_Toc486841776)

[11 Boot Paths 22](#_Toc486841777)

[11.1 Introduction 22](#_Toc486841778)

[11.2 Code Flow 22](#_Toc486841779)

[11.3 Normal Boot Path 22](#_Toc486841780)

[11.4 Recovery Path（BOOT\_IN\_RECOVERY\_MODE） 22](#_Toc486841781)

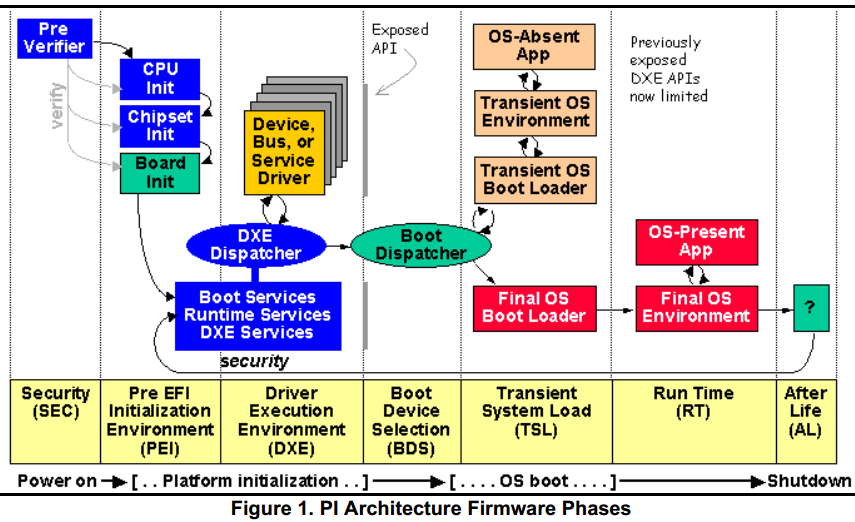
[11.5 Priority of Boot Path 22](#_Toc486841782)

[11.6 Boot Path约定 23](#_Toc486841783)

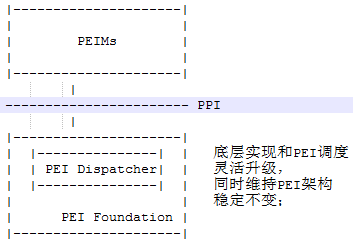
[12 PEI Physical Memory Usage 23](#_Toc486841784)

[13 TE（Terse Execution）Image、Create、Loading 24](#_Toc486841785)

1. PI Architecture firmware Phase



1. SEC
   1. Cache As RAM（CAR）、stack location、BFV location（通过EFI\_SEC\_PEI\_HAND\_OFF传给PEI）
   2. 加载MicroCode
   3. 从Boot Firmware Volume中discover PEI Foundation，authenticate it，并把控制权交给PEI Foundation（Boot Firmware Volume也被称为FV0，FV0定义为PEI Foundation所在的Firmware Volume，FV0用来获取其它Firmware Volume的地址）。
   4. FV0中的其它file 将在PEI阶段discover and validate。
2. PEI Overview
   1. **Introduction**
      1. 存在的主要目的是discover and initialize **Main Memory**
      2. 本质上是微型版的DXE。主要区别是（1）绝大部分PEIM的运行环境是BIOS Flash；（2）PEI阶段**后期**才有**少量**内存；（3）仅个别PEIM（如DXE IPL PPI、crisis recovery，这些PEIM一般会被压缩）的运行环境是Main Memory。
      3. 特点：Nascent state（Size小、慢速、功能少）。
      4. Dispatcher PEIM：crisis recovery, Resume from S3。
   2. PEI架构



* 1. **Goal**
     1. 为DXE初始化内存
     2. 保证“chain of trust”
     3. PEI Foundation（Core PEI modules），通过PEI modules可以支持不同厂商CPU、Chipsets、memory、SIO等初始化
     4. 自由灵活的PEI modules（PEIM）
  2. **PEI consists of**（见Figure1）
     1. PEI foundation
        1. Load、validate、dispatch PEIM
        2. Facilitate communication with PEIM（PPI）
        3. Provide handoff data to DXE
     2. PEIM（plug-in）

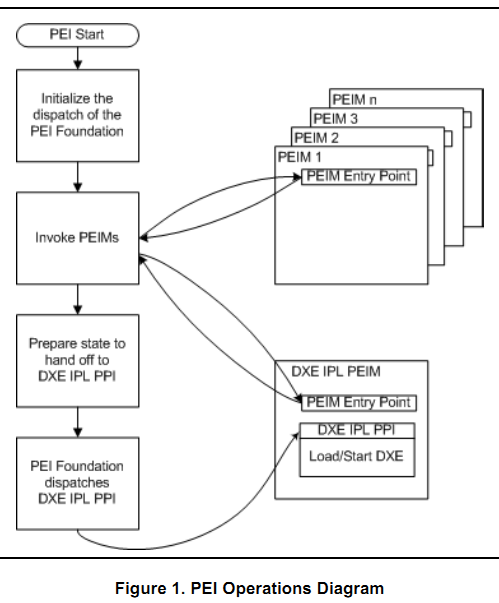


Figure 1 PEI Operation

* 1. **PEI Service Table**（**struct** \_EFI\_PEI\_SERVICES），**not rich** because of philosophy of PEI phase
     1. PPI FUNCTIONS
        1. InstallPpi（install EFI\_PEI\_PPI\_DESCRIPTOR into PPI database by GUID, and publish it to PEIM）、ReInstallPpi
        2. LocatePpi
        3. NotifyPpi（EFI\_PEI\_NOTIFY\_DESCRIPTOR：Notify某功能<EFI\_PEIM\_NOTIFY\_ENTRY\_POINT> when指定的PPI<由Guid指定> installed）
     2. Boot Mode Functions
        1. GetBootMode
        2. SetBootMode

**#define** BOOT\_WITH\_FULL\_CONFIGURATION 0x00

**#define** BOOT\_WITH\_MINIMAL\_CONFIGURATION 0x01

**#define** BOOT\_ASSUMING\_NO\_CONFIGURATION\_CHANGES 0x02

**#define** BOOT\_WITH\_FULL\_CONFIGURATION\_PLUS\_DIAGNOSTICS 0x03

**#define** BOOT\_WITH\_DEFAULT\_SETTINGS 0x04

**#define** BOOT\_ON\_S4\_RESUME 0x05

**#define** BOOT\_ON\_S5\_RESUME 0x06

**#define** BOOT\_WITH\_MFG\_MODE\_SETTINGS 0x07

**#define** BOOT\_ON\_S2\_RESUME 0x10

**#define** BOOT\_ON\_S3\_RESUME 0x11

**#define** BOOT\_ON\_FLASH\_UPDATE 0x12

**#define** BOOT\_IN\_RECOVERY\_MODE 0x20

* + 1. HOB Functions
       1. GetHobList
       2. CreateHob(变量<StatusCode、OC\_WDT>、函数指针、PEIM入口、内存空间)

**#define** EFI\_HOB\_TYPE\_HANDOFF 0x0001

**#define** EFI\_HOB\_TYPE\_MEMORY\_ALLOCATION 0x0002（内存空间）

**#define** EFI\_HOB\_TYPE\_RESOURCE\_DESCRIPTOR 0x0003（内存空间）

**#define** EFI\_HOB\_TYPE\_GUID\_EXTENSION 0x0004（变量/PEIM入口…）

**#define** EFI\_HOB\_TYPE\_FV 0x0005

**#define** EFI\_HOB\_TYPE\_CPU 0x0006

**#define** EFI\_HOB\_TYPE\_MEMORY\_POOL 0x0007

**#define** EFI\_HOB\_TYPE\_FV2 0x0009

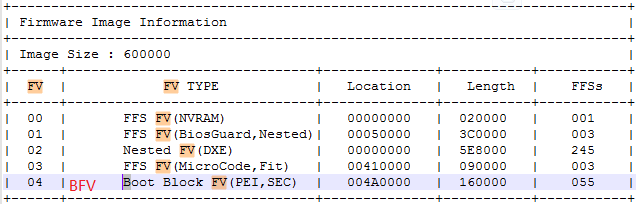
**#define** EFI\_HOB\_TYPE\_LOAD\_PEIM\_UNUSED 0x000A

**#define** EFI\_HOB\_TYPE\_UEFI\_CAPSULE 0x000B

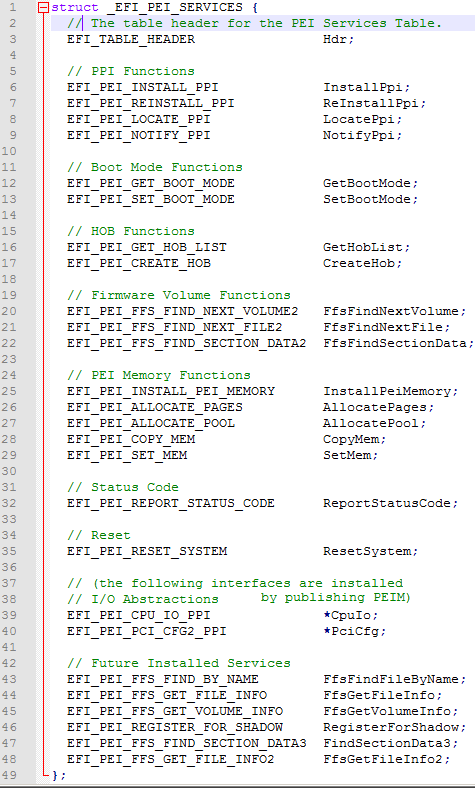
**#define** EFI\_HOB\_TYPE\_UNUSED 0xFFFE

**#define** EFI\_HOB\_TYPE\_END\_OF\_HOB\_LIST 0xFFFF

* + 1. Firmware Volume Functions
       1. FfsFindNextVolume
       2. FfsFindNextFile
       3. FfsFindSectionData
    2. PEI Memory Functions
       1. InstallPeiMemory
       2. AllocatePages
       3. AllocatePool
       4. CopyMem
       5. SetMem
    3. Status Code
       1. ReportStatusCode
          1. Code type：EFI\_**PROGRESS**\_CODE、EFI\_**ERROR**\_CODE（EFI\_ERROR\_MINOR、MAJOR、UNRECOVERD、UNCONTAINED）、EFI\_**DEBUG**\_CODE。
          2. Code Value：
    4. Reset
       1. ResetSystem (reset processor and all device, and reboot)。
    5. I/O Abstractions
       1. CpuIo
       2. PciCfg
    6. Future Installed Services
  1. **PEI Foundation（被设计成硬件平台无关）的工作内容**
     1. Dispatch PEIM
     2. Maintain boot mode（见1.2.4.2）
     3. Init memory
     4. Invok DXE loader（DXE IPL PPI）
  2. **PEI Dispacher（状态机）**
     1. PEI Foundation把控制权交给PEI Dispatcher。
     2. PEI Dispatcher从BFV顺序搜索type为EFI\_FV\_FILETYPE\_PEIM和EFI\_FV\_FILETYPE\_CONBINED\_PEIM\_DRIVER的file，并检查PEIM的依赖关系（根据PEI Foundation里PPI database），发现某PEIM满足依赖关系（即dependency expression为True），并汇报给PEI Foundation。
     3. PEI Foundation把控制权由交给该PEIM执行，执行完PEIM后再将控制权交回PEI Dispatcher。
     4. 重复1、2、3，直到“PEI Dispatcher检查完firmware里的所有PEIM，再没找到可被执行PEIM（满足dependency expression）”，并汇报给PEI Foundation。
     5. PEI foundation重新从PEI Dispatcher获得控制权，并把控制权交给DXE Loader（DXE IPL PPI），从此结束PEI阶段，进入DXE。
  3. **PEIM**
     1. BIOS Flash中执行，poor memory
     2. PEI越小越好，如果可能那么工作最好放在DXE Driver里。
  4. **PPI（PEIM间的通信）**
     1. 如果某PEIM要提供服务给其他PEIM，就InstallPpi或ReinstallPpi，NotifyPpi（notify某功能，when 某ppi installed）。
     2. 如果某PEIM要使用其他PEIM提供的服务，就LocatePpi。
     3. 还有两类PPI：Architecture PPI、Additional PPI。
  5. **Firmware Volume -> Firmware File System -> Data Section**
     1. PEI Foundation必须在BFV中。
     2. 下图是SkylakeDT 6MB BIOS文件FV布局。



1. PEI Service Table（PEI Foundation创建；services are not rich；使用PEIM和PPI，须传递“IN VOID \*PeiServices”指针）。



1. PEI Service
   1. PPI FUNCTIONS
      1. PEI Foundation维护一个对PEIM透明的PPI Database（单向链表，通过GUID和instance唯一定位某PPI），并通过允许PEIM使用如下service安装、搜寻PPI。
      2. InstallPpi（install EFI\_PEI\_PPI\_DESCRIPTOR into PPI database by GUID, and publish it to PEIM）、ReInstallPpi
      3. LocatePpi：通过GUID和instance唯一定位某PPI。
      4. NotifyPpi（EFI\_PEI\_NOTIFY\_DESCRIPTOR：install

notification service

“某功能<EFI\_PEIM\_NOTIFY\_ENTRY\_POINT>”

to be callback，when **Guid指定的PPI** is installed）。

* 1. Boot Mode Functions
     1. GetBootMode
     2. SetBootMode

**#define** BOOT\_WITH\_FULL\_CONFIGURATION 0x00

**#define** BOOT\_WITH\_MINIMAL\_CONFIGURATION 0x01

**#define** BOOT\_ASSUMING\_NO\_CONFIGURATION\_CHANGES 0x02

**#define** BOOT\_WITH\_FULL\_CONFIGURATION\_PLUS\_DIAGNOSTICS 0x03

**#define** BOOT\_WITH\_DEFAULT\_SETTINGS 0x04

**#define** BOOT\_ON\_S4\_RESUME 0x05

**#define** BOOT\_ON\_S5\_RESUME 0x06

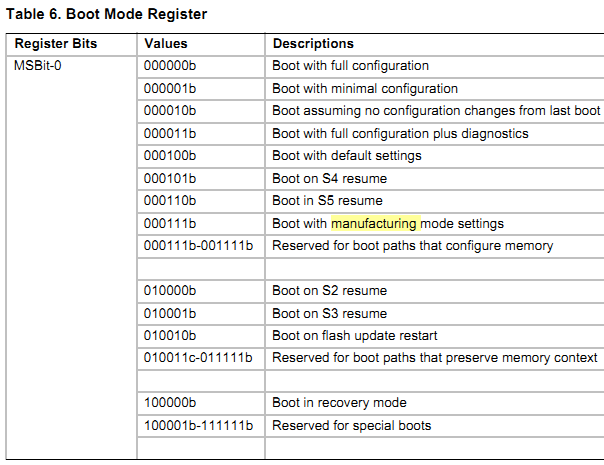
**#define** BOOT\_WITH\_MFG\_MODE\_SETTINGS 0x07

**#define** BOOT\_ON\_S2\_RESUME 0x10

**#define** BOOT\_ON\_S3\_RESUME 0x11

**#define** BOOT\_ON\_FLASH\_UPDATE 0x12

**#define** BOOT\_IN\_RECOVERY\_MODE 0x20



* 1. HOB Functions
     1. 为了向DXE传递信息，PEI Foundation维护Hob list，并通过如下方法操作该Hob list（Hob general header结构体中，<1>用HobLength获取Hob的长度和下一个Hob的地址，<2>用HobType获取当前Hob的类型）；使用**AllocatePool** service从Hob 堆中分配。
     2. GetHobList
     3. CreateHob(变量<StatusCode、OC\_WDT>、函数指针、PEIM入口、内存空间)

**#define** **EFI\_HOB\_TYPE\_HANDOFF** 0x0001

**#define** EFI\_HOB\_TYPE\_MEMORY\_ALLOCATION 0x0002（内存空间）

**#define** EFI\_HOB\_TYPE\_RESOURCE\_DESCRIPTOR 0x0003（内存空间）

**#define** EFI\_HOB\_TYPE\_GUID\_EXTENSION 0x0004（变量/PEIM入口…）

**#define** EFI\_HOB\_TYPE\_FV 0x0005

**#define** EFI\_HOB\_TYPE\_CPU 0x0006

**#define** EFI\_HOB\_TYPE\_MEMORY\_POOL 0x0007

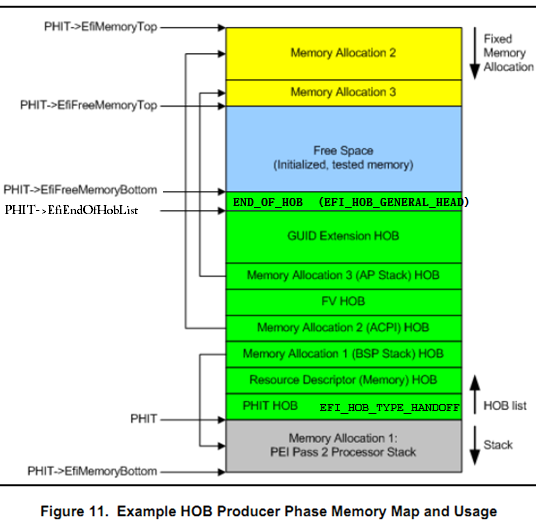
**#define** EFI\_HOB\_TYPE\_FV2 0x0009

**#define** EFI\_HOB\_TYPE\_LOAD\_PEIM\_UNUSED 0x000A

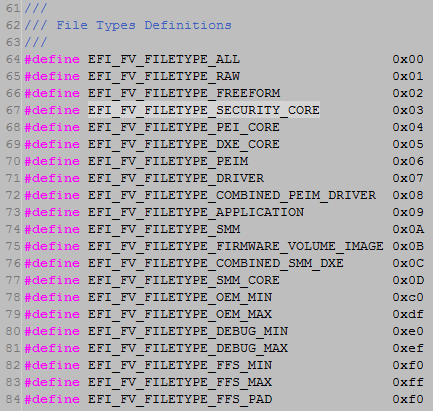
**#define** EFI\_HOB\_TYPE\_UEFI\_CAPSULE 0x000B

**#define** EFI\_HOB\_TYPE\_UNUSED 0xFFFE

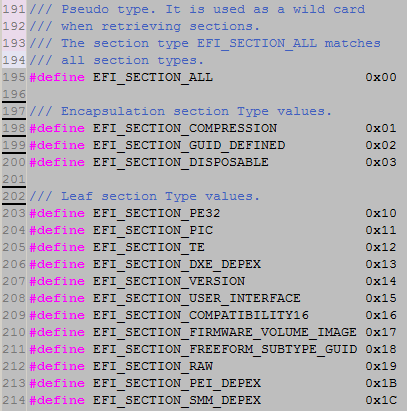
**#define** **EFI\_HOB\_TYPE\_END\_OF\_HOB\_LIST** 0xFFFF



* 1. Firmware Volume Functions
     1. 为允许PEIM访问FV，PEI Foundation提供查找FV、File和SectionData的方法，如下
     2. FfsFindNextVolume
     3. FfsFindNextFile（用VolumeHandle和FileType来依次查找匹配的FileHandle）



* + 1. FfsFindSectionData(通过File Handle和Section Type来查找匹配的Section Data)



* + 1. RegisterForShadow：当前正在执行的PEIM file 将被重新加载到permanent memory，并会再次运行一次。由于dependency expression的缘故，某些被Shadow的PEIM file的两次执行可能都在permanent memory中执行。
  1. PEI Memory Functions
     1. PEI Foundation管理memory，并提供如下service；另外下面service适用于temporary memory和permanent memory。
     2. PEI Foundation初始化permanent memory
     3. InstallPeiMemory：PEIM发现PEI Foundation已初始化的permanent memory后，调用该service注册permanent memory configuration，这样才可以调用AllocatePages、AllocatePool获取memory。Installed的内存一般通过HOB汇报给DXE阶段。
     4. AllocatePages
        1. Recovery PPIs会很喜欢这个service
        2. The type of memory to allocate：
           1. Efi**Loader**Code, EfiLoaderData,
           2. Efi**RuntimeServices**Code, EfiRuntimeServicesData,
           3. Efi**BootServices**Code,EfiBootServicesData,
           4. Efi**ACPIReclaim**Memory, Efi**ACPIMemory**NVS,
           5. Efi**ReservedMemory**Type,
     5. AllocatePool:从HOB堆上获取内存（见1.4.3）
     6. CopyMem：copy to **Arg0** from **Arg1**
     7. SetMem：set **buffer** in **size** with **value**
  2. Status Code
     1. ReportStatusCode
        1. Code type：EFI\_**PROGRESS**\_CODE、EFI\_**ERROR**\_CODE（EFI\_ERROR\_MINOR、MAJOR、UNRECOVERD、UNCONTAINED）、EFI\_**DEBUG**\_CODE。
        2. Code Value：
  3. Reset
     + 1. ResetSystem (reset processor and all device, and reboot)。
  4. I/O Abstractions
     + 1. CpuIo
       2. PciCfg
  5. Future Installed Services

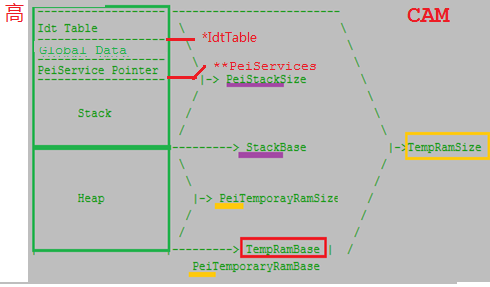
1. PEI Foundation（含Dispatcher）
   1. PEI Foundation entry point
      1. After **Assembly code** part of **SEC** phase 初始化了temporary memory和Stack，SEC phase进入**C code** part of **SEC** phase（代码in SecMain.c），具体：
         1. 未启用FSP\_WRAPPER\_FLAG（Skylake默认）：

* **SecStartup** (IN UINT32 SizeOfRam,

IN UINT32 TempRamBase,

IN VOID \*BootFirmwareVolume)

**为PEI准备BFV、temporaryRam、PeiTemporaryRam、Stack的BaseAddress和Size。**



* **SecStartupPhase2**(

IN EFI\_SEC\_PEI\_HAND\_OFF \*SecCoreData)

生成PpiList，在BFV找到并调用**PeiCoreEntryPoint。**

* (\***PeiCoreEntryPoint**) (SecCoreData, PpiList)
  + - 1. 启用FSP\_WRAPPER\_FLAG：
* **SecFspInit** ( IN FSP\_INFO\_HEADER \*pFspHeader,

IN UINT32 StackTop,

IN UINT32 StackBottom)

* **FspWrapperCallPeiCore** (

IN FSP\_INFO\_HEADER \*pFspHeader,

IN UINT32 StackTop,

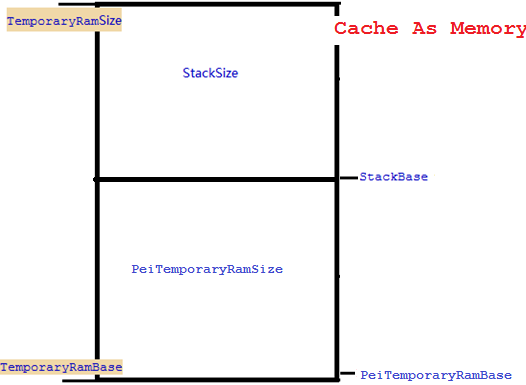
IN UINT32 StackBottom)

* (\***PeiCoreEntryPoint**) (SecCoreData, PpiList)
  + - 1. 控制权交由**PeiCoreEntryPoint**后，正式进入PEI阶段
    1. Typedef VOID EFIAPI (\***EFI\_PEI\_CORE\_ENTRY\_POINT**)(

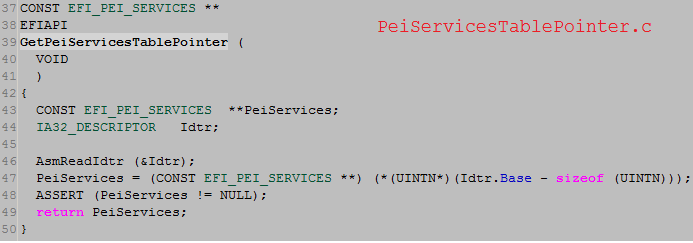
IN CONST EFI\_**SEC\_PEI\_HAND\_OFF** \*SecCoreData,

IN CONST EFI\_PEI\_**PPI\_DESCRIPTOR** \*PpiList)

* + - 1. 函数实体位于PeiCoreEntryPoint.c的**\_ModuleEntryPoint。**
      2. EFI\_**SEC\_PEI\_HAND\_OFF** \*SecCoreData：SEC阶段传给PEI阶段的BFV、temporary RAM、Stack、temporary RAM for PEI foundation的BaseAddress和Size



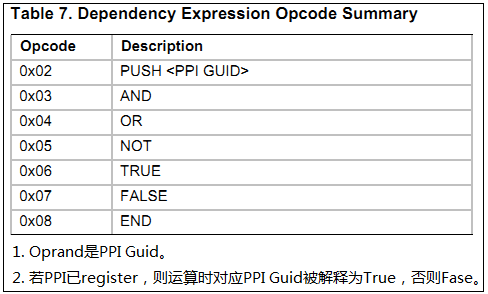
* + - 1. EFI\_PEI\_**PPI\_DESCRIPTOR** \*PpiList：SEC传给PEI的a list of SEC-host PPI descriptor，PEI将把SEC-host PpiList增加到自己初始化的PpiList列表中，这样PEIM和PEIFoundation就可用那些early PPI。
  1. **如何获取 \*\*PeiServices**



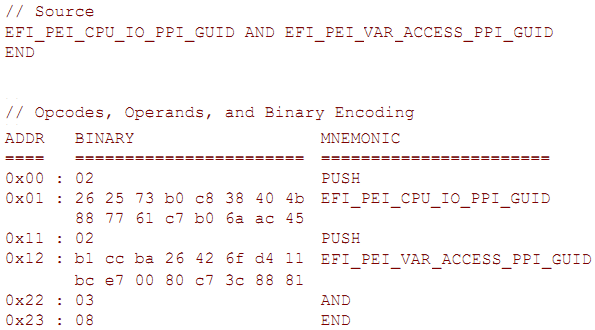
* 1. **Pei Dispatcher Introduction**
     1. PeiDispatcher于FVs**按序**搜type为EFI\_FV\_FILETYPE\_PEIM和

EFI\_FV\_FILETYPE\_CONBINED\_PEIM\_DRIVER的ffs file（PEIM），检查其depex，若depex为True，则把控制权交给该ffs file对应的PEIM；PEIM执行完后把控制权交回PeiDispater；如此循环，直到所有depex为True的ffs file都已执行，PeiDisapter把控制权交给PEI Foundation；PEI Foundation将控制权交给DXE IPL PPI，此后进入DXE阶段。

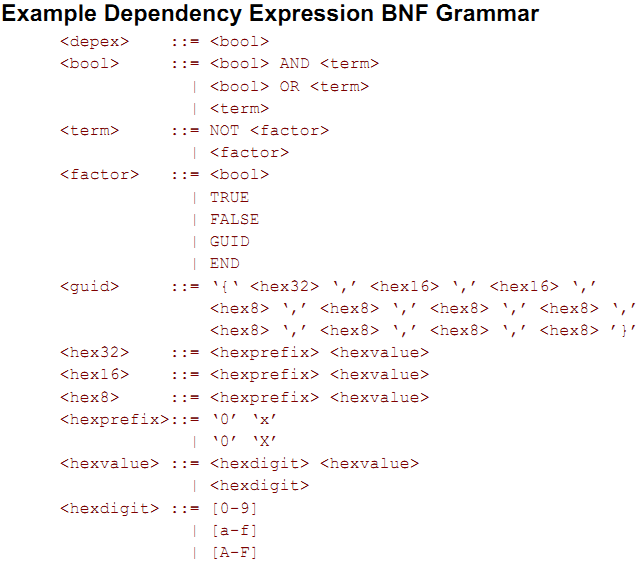
* 1. **PEIM Odering & depex grammar**
     1. 虽然在build阶段，已确定了PEIM执行顺序，但这个顺序不可靠，因为主板外设变化多样，还需要运行状态的PEIM调度机制。
     2. PEI a priori ffs file
        1. 以FV中的一个file形式存在，且每个FV最多一个a priori file。
        2. “bug”设计，PeiDispatcher优先执行“a priori”文件包含的FFS file Guid对应的PEIM。执行完后再根据被称为“weak ordering”的depex来调度PEIM。
     3. PEI depex & depex grammar
        1. 以PEIM（或ffs file）的section形式存在（PEIM 可包含很多section，如dependency section、PE32+ image、Version、RAW等）。
        2. PEI Foundation根据PEIM file的Type找到PEIM（EFI\_FV\_FILETYPE\_PEIM），然后搜索PEIM file中的depex section（section header、 depex<byte stream of Oprands和 Opcodes>）。
        3. 为了节省空间、快速允许，Depex被编码成stack-based instruction set，其OpCodes和OpRands定义如下：



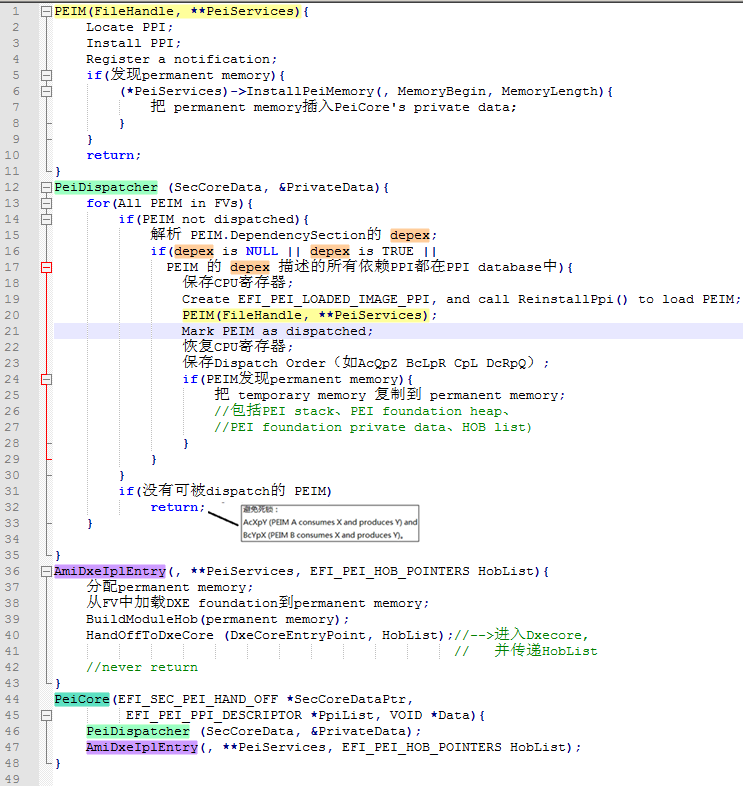
* + - 1. Depex的Binary Code采用逆波兰表达式。



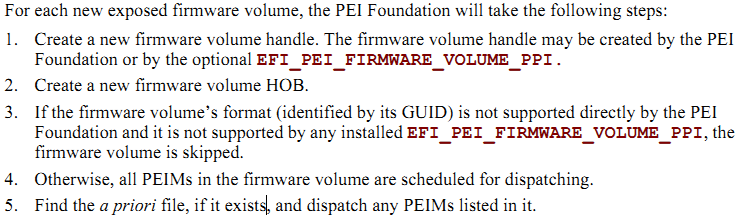
* + - 1. Depex的Source Code采用中缀表达式。



* 1. **Dispatcher Algorithm**
     1. Ordering Algorithm

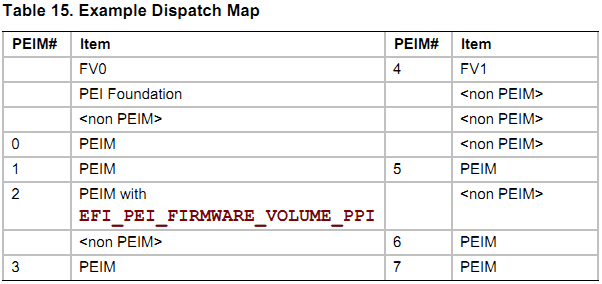


* + 1. Multiple Firmware Volume Support

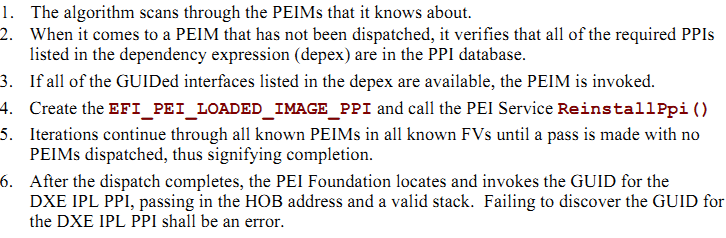


* + 1. Example Dispatch Algorithm

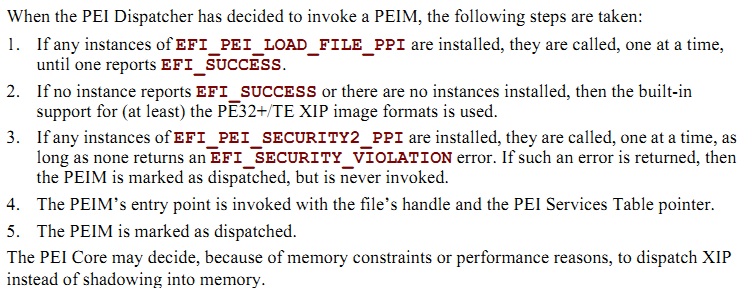
An example of a dispatch in a given set of firmware volumes (FVs).



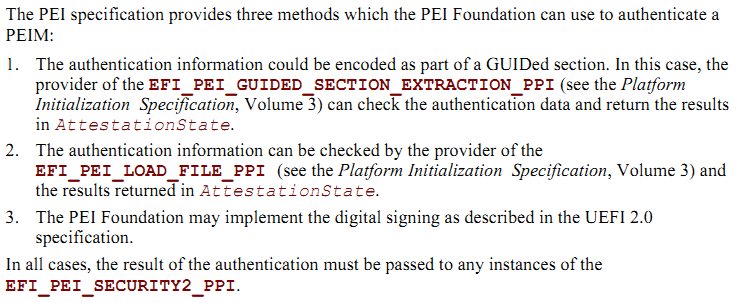
The steps in this dispatch.



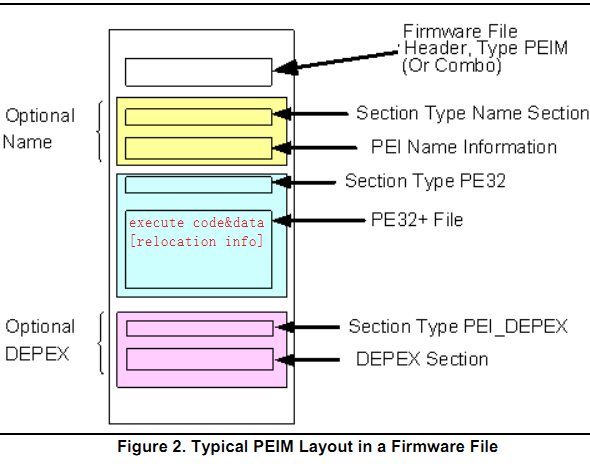
The Steps to dispatch PEIM



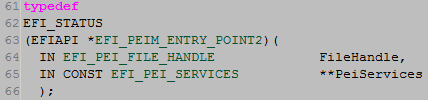
* + 1. PEI Authenticate



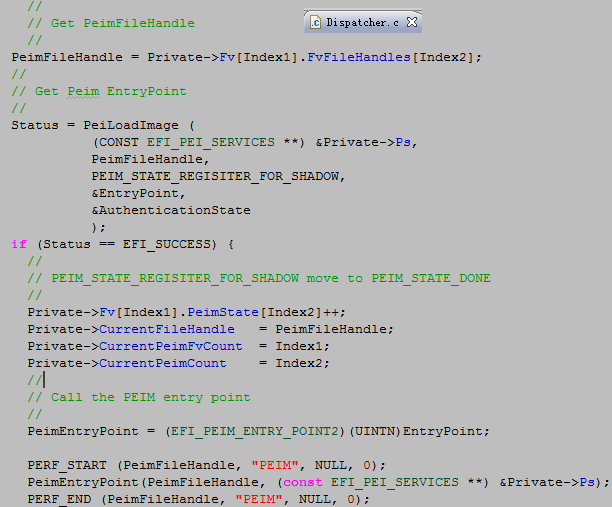
1. PEIMs & PPI的安装与获取
   1. **PEI阶段**，PEIMs是execution unit，而PEI Foundation是execution unit调度器。
   2. **PEIMs间通过PPI通信**，还可使用全局的PeiServices提供的服务。
   3. **与DXE Driver类似，PEIM有5类**：
      1. Processer-specific PEIMs
      2. Chipset-specific PEIMs
      3. Platform-specific PEIMs
      4. PEI-CIS规范描述的architecture PEIMs
      5. Misc PEIMs
   4. **PEIM Structure**
      1. PEIM Header
      2. Execute in place code/data（后期执行的一些压缩的PEIM<recovery PEIM、DXE IPL PPI等>会在内存里执行）
      3. Optional relocation information（若code是position dependent，此时在flash中PEIM代码执行地址与编译时确认的不同< 一般BaseAddress = 0 >；需要FV store software<FV update tool>根据relocation information修改其地址）。
      4. Authentication information Section, if present：
         1. Section类型：EFI\_SECTION\_GUID\_DEFINED
         2. 内容：hash校验、一般校验、CRC校验
         3. image Authentication是基于code在relocation前的image，所以在authentication时必须undo the modified image。



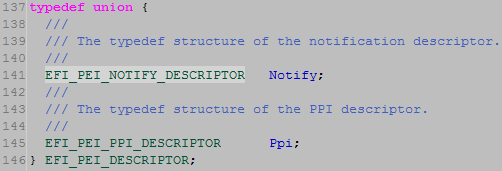
* 1. **PEIM Invocation Entry Point**
     1. PEIM 入口函数类型（EFI\_PEIM\_ENTRY\_POINT）定义：

****

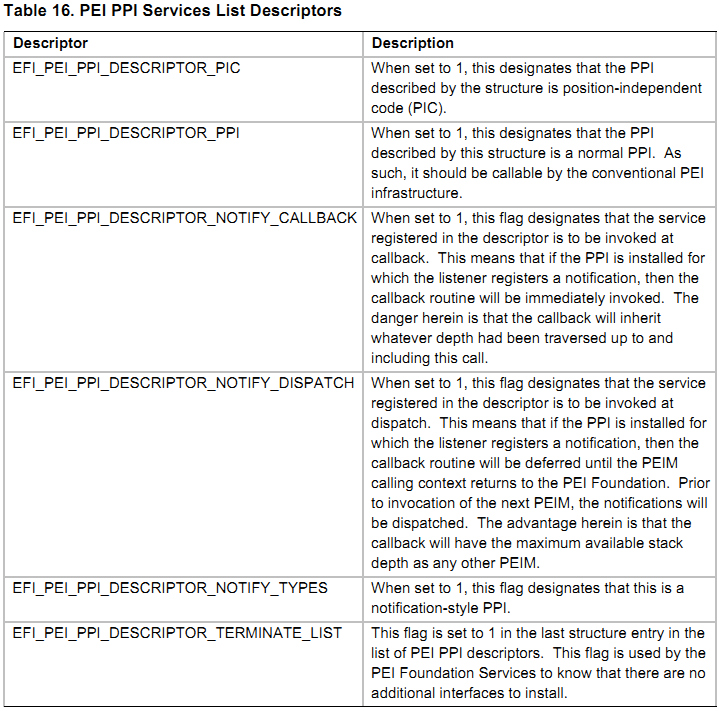
* + 1. PEIM以ffs file形式存储在Image的FV中，FileHandle便是该ffs file的handle。
    2. 如果不考虑RegisterForShadow的PEIM在permanent memory里还会被执行一次（一共执行两次），所有PEIM最多会被PEI Dispatcher调用一次。
    3. 调用加载、调用PEIM的代码：



* + 1. 类似，EFI\_IMAGE\_ENTRY\_POINT是DXE/UEFI Driver入口函数类型
  1. PEIM descriptor（EFI\_PEI\_DESCRIPTOR）
     1. 包括flag、Guid、函数指针（ppi<给其他PEIM用>或**notify function<给PEI Foundation用>**）。

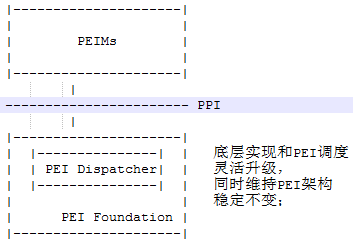


* + 1. 用来输出service给其它PEIM或**PEI Foundation（用来向PEI Foundation请求响应某事件）**。
    2. EFI\_PEI\_**NOTYFY**\_DESCRIPTOR（用来**补充完成depex机制<对无depex，但依然有依赖关系要求的PEIMs>**）
       1. PEIM通过(\*PeiServices)->NotifyPpi(PEISERVICES, NOTIFYLIST)，告诉PEI Foundation“给我xxx PPIs或PEIMs，不给立马罢工”（当然有个前提：被依赖的xxx PPIs被install成callback或dispatch类型，即允许使用者等待）
       2. 当这些PPIs installed/reinstalled或PEIMs已完成，那么根据NOTIFYLIST （EFI\_PEI\_NOTYFY\_DESCRIPTOR）中的**Flag**
* Flag = EFI\_PEI\_PPI\_DESCRIPTOR\_NOTIFY\_CALLBACK，则PEI Foundation立马中断当前PEIM，立马调用callback函数，待该callback函数执行完再回到被中断的PEIM。
* Flag = EFI\_PEI\_PPI\_DESCRIPTOR\_NOTIFY\_DISPATCH，则PEI Foundation等待当前PEIM执行完毕后，通知所有依赖该PPI或PEIMs的PEIMs，你要的我都给了，赶快干活。
  + - 1. 总之，PEIM缺什么就用NotifyPpi手段向PEI Foundation索取。
    1. EFI\_PEI\_PPI\_DESCRIPTOR
       1. PEIM可提供或获取PPI服务（InstallPpi、ReinstallPpi和LocatePpi）。



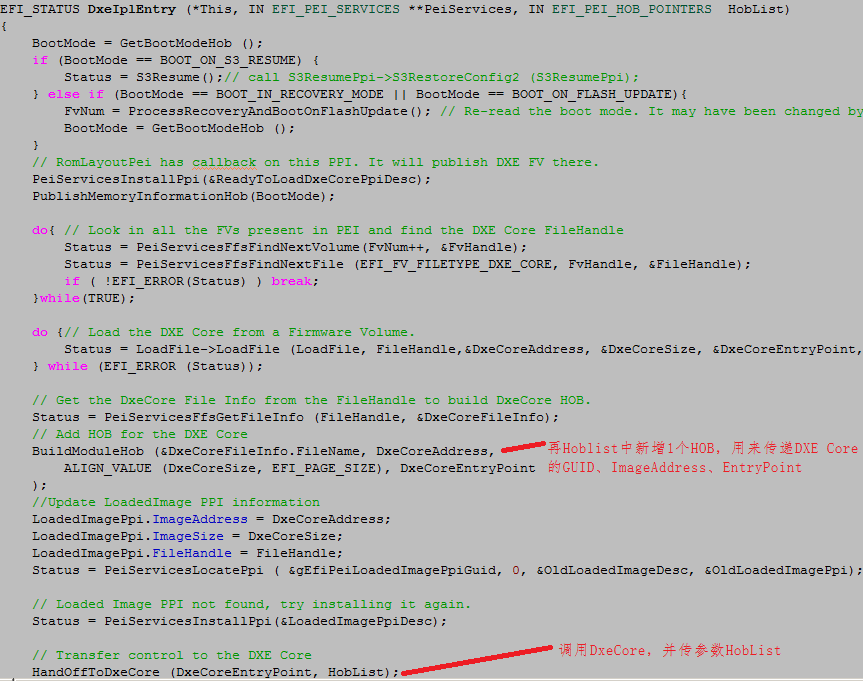
* + 1. PEIM-to-PEIM communication
       1. InstallPpi、ReinstallPpi、LocatePpi，NotifyPpi。

1. Architecture PPI
   1. 一般来讲，Architecture PPI是提供给PEI Foundation用的
   2. PEI Foundation和PEI Dispatcher通过PPI来协调PEIM完成PEI阶段的工作，如下图

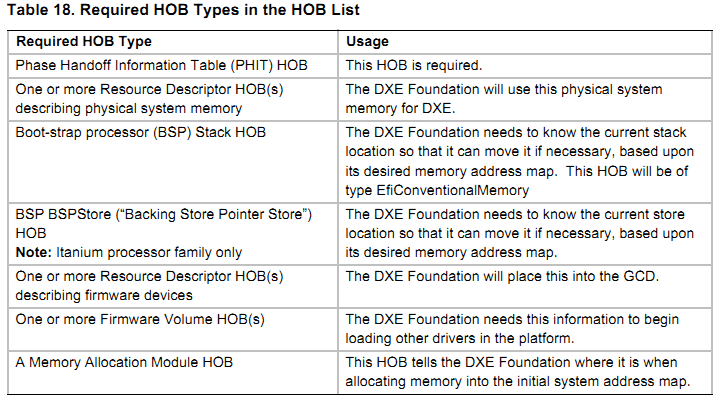


* 1. EFI\_PEI\_MASTER\_BOOT\_MODE\_PPI
  2. EFI\_DXE\_IPL\_PPI（能力）
     1. PEI foundation 最后调用
        1. Locating and invoking DXE Foundation
        2. Invoking recovery processing module
        3. Invoking S3 resume module
  3. EFI\_PEI\_PERMANENT\_MEMORY\_INSTALLED\_PPI**（通知）**
     1. Permanent memory已经registered时，由PEI foundation来install。
     2. 由需要使用permanent memory的PEIM使用。
     3. 其它PPI可依此获知permanent memory已准备好。
  4. EFI\_PEI\_BOOT\_IN\_RECOVERY\_MODE\_PPI**（通知）**
     1. 当platform PEIM认为发生异常需要进入recovery模式时，install 该PPI。
     2. 进入crisis recovery mode。Recovery mode下执行的PEIM须把该PPI加入其depex，这样normal mode下不会加载该PEIM（仅在Recovery mode加载）。
  5. EFI\_PEI\_END\_OF\_PEI\_PHASE\_PPI**（通知）**
     1. Normal mode下，由DXE IPL PPI来install；S3Resume mode，由EFI\_PEI\_S3\_RESUME\_PPI.S2RestoreConfig来install。
     2. 预示着PEI phase将结束，提醒其它PEIM准备后事，比如把内存转换成HOB等。
  6. EFI\_PEI\_RESET\_PPI、EFI\_PEI\_RESET2\_PPI（能力）
     1. 由某些platform-或chipset- specific PEIM给其它PEIM提供reset platform的能力。
  7. EFI\_PEI\_PROGRESS\_CODE\_PPI（能力）
     1. 提供其它PEIM输出status code的能力
  8. EFI\_PEI\_SECURITY\_PPI（能力）
     1. 提供PEI Foundation验证PEIM FFS FILE是否被串改的能力。
  9. EFI\_PEI\_TEMPORARY\_RAM\_SUPPORT\_PPI（能力）
     1. 由SEC或PEIM来install。
     2. 若temporary memory和permanent memory这两种memory同时存在有副作用，则可用该PPI提供给其它PEIM**把temporary memory转换成permanent memory**的能力。
     3. 一般调用InstallPeiMemory后，调用TemoraryRamMigration，再调用PPI EFI\_PEI\_PERMANENT\_MEMORY\_INSTALLED\_PPI 通知其它PEIM，permanent memory已准备就绪。
  10. EFI\_PEI\_TEMPORARY\_RAM\_DONE\_PPI（能力）
      1. 由SEC或PEIM来install。
      2. 提供其它PEIM **disable temporary memory**的能力。

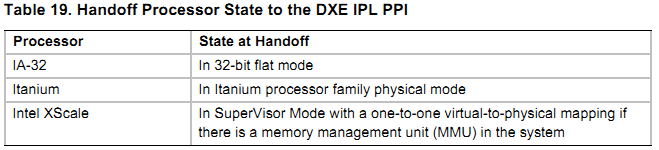
1. Additional PPI
   1. 一般来讲，Additional PPI分类的设计初衷是提供给PEIM使用的。
   2. Required
      1. EFI\_PEI\_PCI\_CFG2\_PPI
      2. EFI\_PEI\_INSTALL\_PPI
      3. EFI\_PEI\_READ\_ONLY\_VARIABLES2\_PPI
   3. Optional
      1. EFI\_**SEC**\_PLAFORM\_INFOMATION\_PPI、EFI\_**SEC**\_PLAFORM\_INFOMATION**2**\_PPI
      2. EFI\_PEI\_LOADED\_IMAGE\_PPI
      3. EFI\_SEC\_HOB\_DATA\_PPI.GetHobs()：允许SEC code 生成HOBs，以用来插入PEI HOB list。
      4. EFI\_PEI\_RECOVERY\_MODULE\_PPI
      5. EFI\_PEI\_DEVICE\_RECOVERY\_MODULE\_PPI
      6. EFI\_PEI\_RECOVERY\_BLOCK\_IO\_PPI
      7. EFI\_PEI\_RECOVERY\_BLOCK\_IO2\_PPI
      8. EFI\_PEI\_VECTOR\_HANDOFF\_INFO\_PPI
      9. **EFI\_PEI\_CPU\_IO\_PPI**
      10. EFI\_PEI\_CAPSULE\_PPI
      11. EFI\_PEI\_CAPSULE\_CHECK\_CAPSULE\_UDPATE
      12. EFI\_MP\_SERVICES\_PPI
      13. EFI\_PEI\_GRAPHICS\_PPI
2. PEI to DXE handoff
   1. Discover and dispatch the DXE Foundation
      1. End of PEI phase, PEI Dispatcher invokes DXE IPL PPI to discover and dispatch DXE CORE.



* 1. HobList必须包含如下类型的HOB



* 1. DXE IPL PPI对CPU状态的要求



1. Boot Paths
   1. Introduction
      1. PEI Foundation没有意识到BootPath的存在，而是由PEIM来GetBootMode()或SetBootMode()。
   2. Code Flow
      1. SEC –> PEI -> DXE -> BDS -> Runtime -> AfterLife
   3. Normal Boot Path

**#define** BOOT\_WITH\_FULL\_CONFIGURATION (S0) 0x00

**#define** BOOT\_WITH\_MINIMAL\_CONFIGURATION (S0) 0x01

**#define** BOOT\_ASSUMING\_NO\_CONFIGURATION\_CHANGES (S0) 0x02

**#define** BOOT\_WITH\_FULL\_CONFIGURATION\_PLUS\_DIAGNOSTICS (S0+诊断) 0x03

**#define** BOOT\_WITH\_DEFAULT\_SETTINGS (S0) 0x04

**#define** BOOT\_ON\_S4\_RESUME 0x05

**#define** BOOT\_ON\_S5\_RESUME 0x06

**#define** BOOT\_WITH\_MFG\_MODE\_SETTINGS 0x07

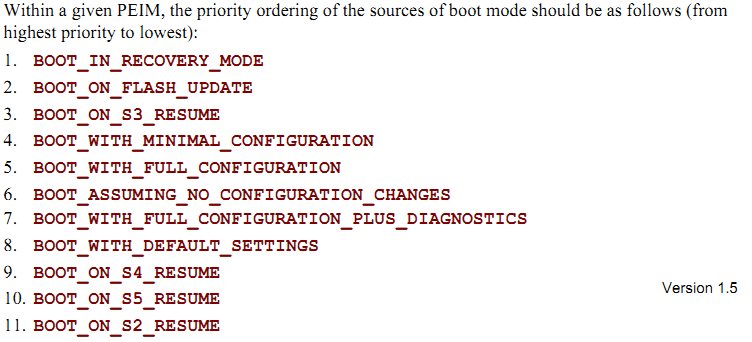
**#define** BOOT\_ON\_S2\_RESUME 0x10

**#define** BOOT\_ON\_S3\_RESUME 0x11

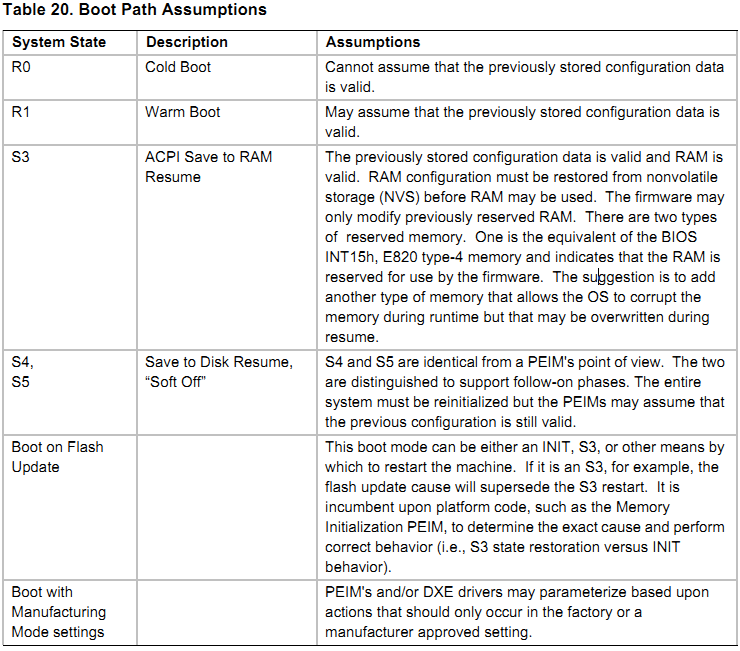
**#define** BOOT\_ON\_FLASH\_UPDATE 0x12

**#define** BOOT\_IN\_RECOVERY\_MODE 0x20

* 1. Recovery Path（BOOT\_IN\_RECOVERY\_MODE）
  2. Priority of Boot Path



* 1. Boot Path约定



1. PEI Physical Memory Usage
   1. Before permanent memory installed
   2. After permanent memory installed
2. TE（Terse Execution）Image、Create、Loading