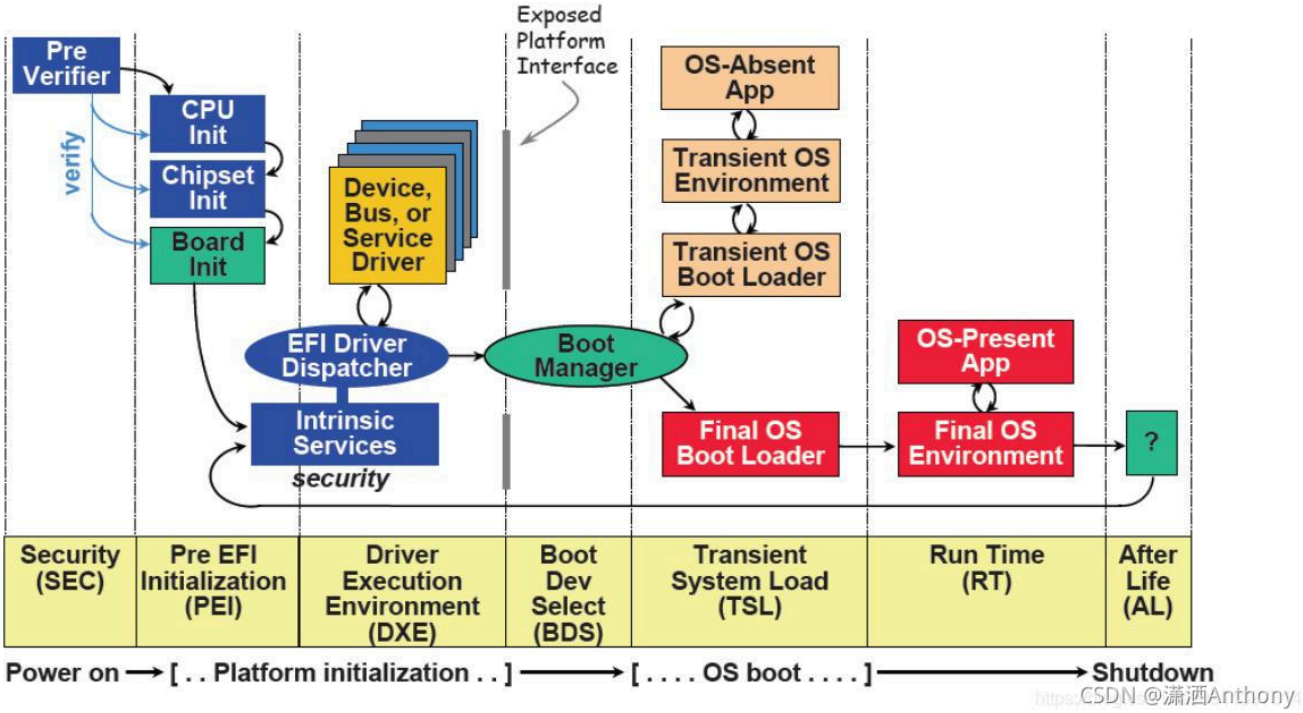
UEFI Principles and Programming Practice--UEFI System Boot Process

原创 Anthony Modified on 2022-02-16 09:43:32 Read 4k Collection 20 Likes

Category Column: UEFI Article Tags: UEFI Start the process

摘要 This article details the boot process of UEFI firmware from the SEC security verification stage to the BDS boot device selection stage, involving key steps such as assembly language, PEI, DXE, BDS, etc. In the code tracing section, special attention is paid to the transition from assembly to C language on the ARM platform, as well as the process of memory initialization, driver execution, and system initialization. The article also proposes the idea of implementing a custom postal code to assist in diag…

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>



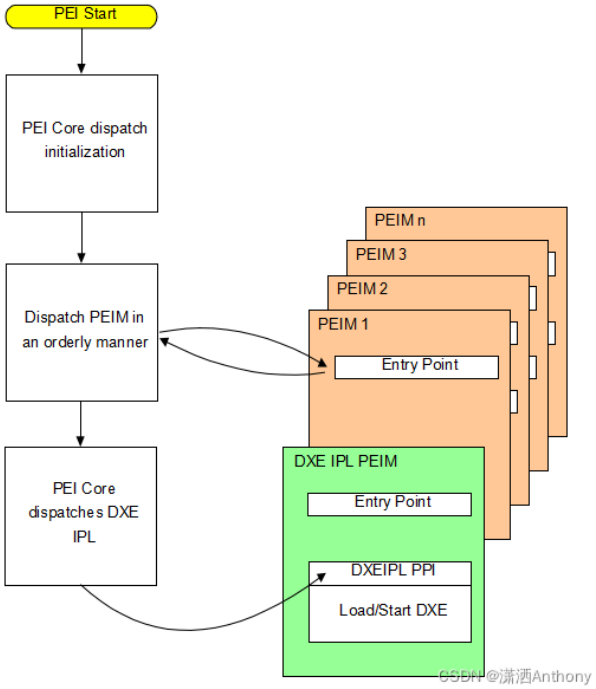
Let's look at the picture and briefly talk about it:

SEC:

Security verification phase, in this phase, Assembly is needed to do some work that C cannot handle, because C language cannot handle special registers of CPU . Let CPU enter Protected Mode (Flat Mode) environment, and use temporary RAM inside CPU, which is actually cache. Assembly is mainly used in this phase, that is to say, establish a compilation environment for C language and hardware communication for the later phase, and finally hand over control to PEI Phase.

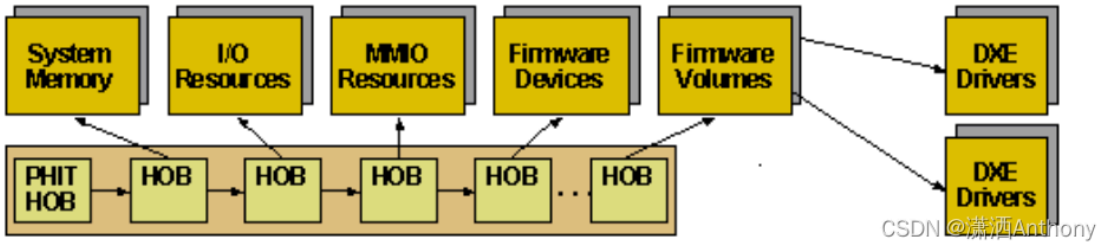
PEI:

Very basic Chipset initialization, Memory Sizing, BIOS Recovery, ACPI S3 Resume, switch Stack to Memory, start DxeIpl. This stage is to start some CPU, motherboard, chip initialization, that is, EFI early initialization, and the memory initialization is in the later stage of this stage. Knowing where the memory is initialized can facilitate debugging. The initialization of the system in the PEI stage is mainly completed by PEIM, and the communication between PEIMs is completed through PPI. The HOB list is required to enter the DXE stage.



HOB : Hand off Block

Some information needs to be passed from PEI Phase to DXE Phase. Each Block has its own GUID & Structure. The Block List of HOB is dynamic and has no order requirement.



DXE:

Traverse all DXE drivers in the firmware, that is, the driver execution environment. Of course, there is also a DXE dispatcher in the DXE stage, and the communication is through protocol. When all DXE drivers are loaded, the system is initialized, DXE finds BDS through EFI_BDS_ARCH_PROTOCOL and calls the entry function of BDS, thus entering the BDS stage. In essence, BDS is a special application in the DXE stage.

BDS:

BDS: Boot Device Selection, starts the necessary drivers and boots the device selection. The BDS policy is configured through the global NVRAM variable, and the boot order (bootorder) is modified by modifying the variable value. This modification of the boot order is often used in factories. Factories are all batch production, so it is necessary to write a tool for the production line TE to batch modify the boot order.

TSL:

The first stage of the operating system loader execution. In this stage, the OS Loader runs as a UEFI application and system resources are still controlled by the UEFI kernel.

RT:

At Run Time, system control is transferred from the UEFI kernel to the OS Loader.

The above was written a long time ago and copied directly. Looking back now, it is actually very general. This time, let's trace the code and see how it runs:

Code Tracing

Since I'm currently using a made in china cpu (ARM), I'll only do some code tracing for this platform.

First, SEC. I won't go into detail about the assembly code here. After SEC is finished, it will enter C. The assembly load is the address of this C entry. After entering the PEI stage, there is a distinction between the early and late stages. Look at the code:

```
VOID
CEntryPoint (
  IN  UINTN                     MpId,
  IN  EFI_PEI_CORE_ENTRY_POINT  PeiCoreEntryPoint
  )
```

```
{
  CHAR8 Char;
  // Data Cache enabled on Primary core when MMU is enabled.
  //ArmDisableDataCache ();
  // Invalidate Data cache
  //ArmInvalidateDataCache ();
  // Invalidate instruction cache
  //ArmInvalidateInstructionCache ();
  // Enable Instruction Caches on all cores.
  Char = 'A';
  SerialPortWrite ((UINT8 *)&Char, 1);
  ArmEnableInstructionCache ();
  Char = 'B';
  SerialPortWrite ((UINT8 *)&Char, 1);
```

收起 ^

The code is in ArmPlatformPkg/PrePeiCore/PrePeiCore.c. Run and run, and finally run to

AI generated projects 登录复制

```
 // Goto primary Main.
    PrimaryMain (PeiCoreEntryPoint);
```

AI generated projects 登录复制

```
// Jump to PEI core entry point
  DEBUG((EFI_D_INFO, "%a() Line=%d \n", __FUNCTION__, __LINE__));
  (PeiCoreEntryPoint)(&SecCoreData, PpiList);
```

Isn't this PeiCoreEntryPoint the entry function of PEI image? _ModuleEntryPoint finally calls the function PeiCore in MdeModulePkg\Core\Pei\PeiMain.c. After entering PeiCore, first set PEI CORE services according to the information passed by SEC

AI generated projects 登录复制

```
 //
  // Initialize PEI Core Services
  //
  InitializeMemoryServices   (&PrivateData, SecCoreData, OldCoreData);
```

Then call PeiDispatcher to execute the PEIM in the system. This process requires the use of the PPI protocol. The process of scheduling the PEIM in the system provides CPU-related functions, such as Cache settings, main frequency settings, memory initialization, IO controller, memory controller, etc. This part is implemented in PlatformPei.efi. By pulling the content in PBF and running MemoryInit.efi after pulling, when the memory is initialized, the system will re-stack and enter PeiCore for the second time. The memory used after re-entering is the memory we are familiar with, and then run CpuPei.efi, and finally prepare the HOB list, use DXE IPL PPI to find the entry function of DXE image, execute the entry function and pass the hob list to DXE, which is mainly run in DxeIpl.efi. After running, it enters DxeCore.efi and finally enters BdsDxe through gBds->Entry (gBds).

The reason why I wrote so much is mainly because I want to make a postcode myself. Writing it here is also a note, especially when the startup interface hangs, I can still find the problem. I think this idea is very good.

AI generated projects 登录复制

```
UINT32
EFIAPI
PostCode (
  IN UINT32  Value
  )
{
  Value &= 0xFFF;
  //Get the StatusCode set using GetPhaseStatusCode
  Value |= GetPhaseStatusCode();
  IoWrite32 (0x80, Value);
  DEBUG ((DEBUG_INFO, "POSTCODE=<%08x>\n", Value));

  return Value;
}
```

收起 ^

Ok, let's stop here for the startup process. . . .