UEFI Development Exploration 66- YIE001PCIe Development Board (02 UEFI Driver)

原创 luobing4365 Posted on 2021-01-13 22:42:03 Read 1.6k Collection 5 Likes 2 copyright

Category Column: UEFI Development Article Tags: uefi bios Low-level application development UEFI Drivers EDK2

UEFI Development This column includes this content 503 Subscribe 104 articles Subscribe to our column

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

In the previous chapters, we have introduced some UEFI driver content. The PCIE Option ROM we want to implement on YIE001 is actually a UEFI driver. This is completely different from the Option ROM of Legacy BIOS. In the 34th blog of the series, we can know that the Legacy Option ROM is completely parallel to the BIOS.

Of course, this description of Legacy Option ROM is not accurate. Legacy BIOS and Legacy Option ROM just follow a calling standard and do not interfere with each other. UEFI Option ROM fully follows the UEFI driver model and can even be used to install Protocol for use by other programs.

Before actually programming, you must clarify the structure and writing method of UEFI driver. It is expected to use about 5 articles to introduce these contents clearly to facilitate the debugging and testing of YIE001 code.

1 UEFI driver classification

From the perspective of type, UEFI drivers can be divided into boot service drivers and runtime drivers. The difference between the two is that after the OS Loader obtains platform control in the RT stage of UEFI BIOS, the runtime driver is still valid.

From a functional perspective, UEFI drivers can be divided into the following categories.

1) Drivers that conform to the UEFI Driver Model. These drivers include bus drivers, device drivers, and hybrid drivers, and are generally used to drive corresponding hardware devices;

2) Service Drivers. This type of driver does not manage any device and is generally used to generate protocols;

3) Initializing Drivers. It does not generate any handles, nor does it add any protocols to the system database. It is mainly used to perform some initialization operations and will be unloaded from the system memory after execution;

4) Root Bridge Drivers. They are used to initialize the root bridge controller on the platform and generate a device address protocol and a protocol for accessing bus devices. They are generally used by bus drivers to access devices. In Section 7.1, we used EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL, which supports access to PCI/PCIe devices, as a typical example.

Drivers that conform to the UEFI driver model and service-type drivers are more commonly used in actual projects, while the other two are less used. This article mainly introduces service-type drivers.

2Service -driven functions

In the previous exploration, we used a lot of Protocol to build various applications. We also learned the basic operation of Protocol in some chapters, especially in 42 and 43, we built a complete service-oriented driver and test program.

The service driver introduced in this article is not much different from the previous ones. It is mainly for the convenience of future reading. In the name of YIE001, the service driver and other contents of UEFI driver are sorted together. Without further ado, let's get into the main article.

Protocol is the most important concept in UEFI. Its understanding and use runs through the entire UEFI exploration blog series. Protocol is provided by various drivers. Among them, service drivers do not need to follow the UEFI driver model, nor do they need to manage any hardware devices. Its main purpose is to generate one or more Protocols and install them on the corresponding service handles.

The Protocol consists of a 128-bit globally unique ID (GUID) and a Protocol interface structure . The structure contains the Protocol interface functions that can be used to access the corresponding device. In the UEFI system, its handle database maintains the relationship between the device handle, Protocol interface, mirror handle, and controller handle. The addition, removal, or replacement of the Protocol interface can be tracked in the handle database, as shown in Figure 1.

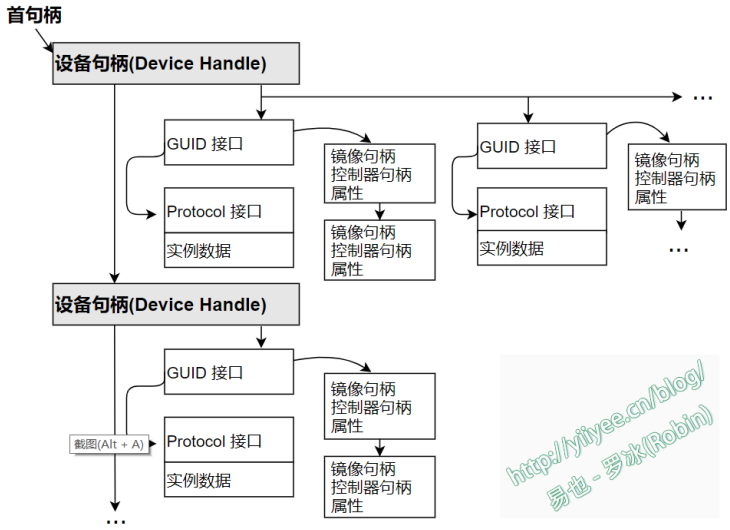

Figure 1 Handle database

In the previous chapter, we have given the functions for processing Protocol in the startup service, which are mainly divided into interface functions for using Protocol and generating Protocol (UEFI Development Exploration 42). The InstallProcotocInterface() and InstallMultipleProtocolInterfaces() functions are used to install a single or multiple Protocols to the device controller; Protocols.

The function to reinstall the Protocol interface is ReinstallProtocolInterface(), which is usually used when the device is replaced, the device path is changed or updated. For example, the UEFI driver that generates the network protocol needs to use this function when using the StationAddress() interface function to modify the MAC address of the network interface.

3. Service-driven framework code construction

The implemented code has made some changes to the example in Chapter 43 and is named ServiceDrv.

Implementing a service-oriented driver framework mainly involves modifying the INF file and installing the sample protocol. Modifying the INF file mainly includes:

1) In the [Defines] section of the INF file, set MODULE_TYPE to UEFI_DRIVER or DXE_DRIVER;

2) Change BASE_NAME under [Defines] Section to the main function entry ServiceDrv of the sample project;

3) Add UefiDriverEntryPoint to [LibraryClasses] Section.

In order to install the protocol we built, that is, EFI_MYSAMPLE_PROTOCOL, we first need to initialize the interface function corresponding to the protocol. Since private data is needed in the example, the protocol instance to be initialized directly uses the member variable myProtocol of the MY_PRIVATE_DATA type global variable gMyData, as shown in Example 1.

[Example 1] Initialize the Protocol to be installed

```
EFI_STATUS MySampleProtocolInit(VOID)
{    MY_PRIVATE_DATA *mydata =&gMyData;      mydata->Signature = MY_PRIVATE_DATA_SIGNATURE;    mydata->myProtocol.Revision=0x101;
//Protocol version    mydata->myProtocol.MySample_In=MySample_In; //The first interface function    mydata->myProtocol.MySample_Out=MySample_Out;
//The second interface function    mydata->myProtocol.MySample_DoSth=MySample_DoSth; //The third interface function return EFI_SUCCESS }
```

After completing the initialization work, you can use the previously introduced service startup interface function InstallProtocolInterface() to install EFI_MYSAMPLE_PROTOCOL. As shown in Example 2, the Protocol is installed on the driver's ImageHandle.

[Example 2] Installing Protocol

```
EFI_STATUS EFIAPI MyProtocolEntry (
  IN EFI_HANDLE ImageHandle,
  IN EFI_SYSTEM_TABLE *SystemTable
  )
{
EFI_STATUS Status;
  MySampleProtocolInit();
Status = gBS->InstallProtocolInterface (
          &ImageHandle, //Image handle
          &gEfiMYSampleProtocolGUID ,//Protocol GUID
          EFI_NATIVE_INTERFACE, //Interface type
          &gMyData.myProtocol //Installed Protocol instance
```

```
        );
  return Status;
}
```

4 Compilation

The compilation method of UEFI driver is the same as that of UEFI application. The sample project ServiceDrv in this section uses the package RobinPkg for compilation. First, add the compilation path in the [Components] Section of RobinPkg.dsc:

RobinPkg/Drivers/ServiceDrv/ServiceDrv.inf

Then start VS2015 x86 Native Tools Command Prompt, enter the EDK2 working directory, and execute edksetup.bat. After starting the UEFI compilation environment, run the following command to compile the IA32 UEFI driver.

```
C:\UEFIWorkspace>build -t VS2015x86 -p RobinPkg\RobinPkg.dsc \
-m RobinPkg\Drivers\ServiceDrv\ServiceDrv.inf -a IA32
```

The compiled target program is also a file in efi format. However, UEFI drivers cannot be run directly in the simulator like UEFI applications, and must be loaded with the help of UEFI Shell commands.

How to test service-oriented drivers and how to use the Protocol provided by service-oriented drivers will be discussed in the next article.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: /FF RobinPkg/ RobinPkg /Drivers/ServiceDrv

about Us Careers Business Cooperation Seeking coverage 400-660-0108 kefu@csdn.net Online Customer Service Working hours 8:30-22:00
Public Security Registration Number 11010502030143 Beijing ICP No. 19004658 Beijing Internet Publishing House [2020] No. 1039-165
Commercial website registration information Beijing Internet Illegal and Harmful Information Reporting Center Parental Control
Online 110 Alarm Service China Internet Reporting Center Chrome Store Download Account Management Specifications
Copyright and Disclaimer Copyright Complaints Publication License Business license
©1999-2025 Beijing Innovation Lezhi Network Technology Co., Ltd.