

UEFI Driver for UEFI

原创

Chisel the wall to borrow light

Posted on 2023-09-14 16:13:25

Read 2.2k

Collection 11

Likes 2

Category Column:


UEFI/BIOS basics

Article Tags:

server

Driver Development


Copyright CC 4.0 BY-SA

UEFI/BIOS basics

This column includes this content

6 articles

Subscribe to our column

摘要

This article introduces the role of UEFI Driver in the computer boot process and how its modular design improves firmware scalability and cross-platform. The article explains in detail the classification of UEFI Driver, including UEFI Driver Model Driver and UEFI Non Driver Model Driver, and uses DiskIoDxeDriver as an example to illustrate the working principle of UEFI Driver Binding Protocol.

The summary is generated in [C Know](#) , supported by DeepSeek-R1 full version, [go to experience>](#)

Table of Contents

- What is UEFI Driver
- UEFI Driver Model
- Classification of UEFI Drivers
- Definition of UEFI Driver
- UEFI Driver Binding Protocol
- DiskIoDxe Driver Example

What is UEFI Driver

UEFI Driver is a special type of driver that is used to load and initialize the software components of hardware devices, such as graphics cards, network cards, sound cards, etc., when the computer starts. Driver is a software program used to control hardware devices. It can run in the operating system and interact with the operating system to communicate with the hardware devices. In simple terms, **UEFI driver is loaded when the computer starts, while Driver is loaded in the operating system.**

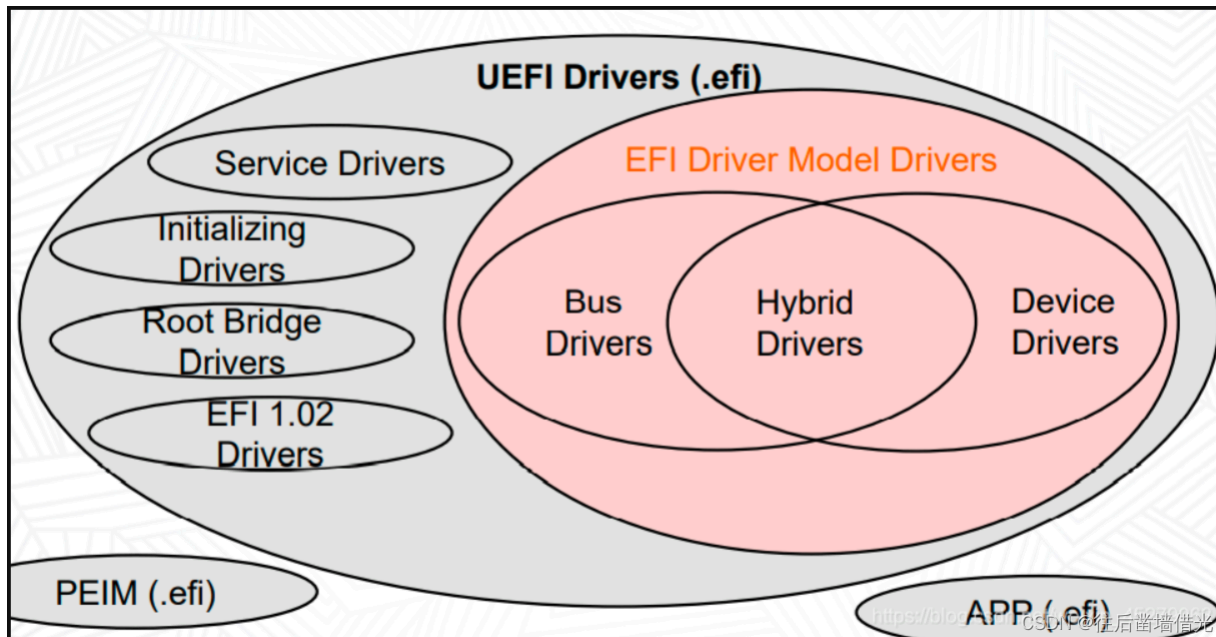
UEFI Driver Model

The introduction of UEFI Driver has better realized modularization. Modularity can be understood as these UEFI Drivers are used to manage devices. If the Driver is used to manage graphics cards, then the graphics card manufacturer can independently write a UEFI Driver, and this Driver can run in all UEFI environments without having to be adjusted according to different environments.

Therefore, for the modularization of UEFI Driver, it can be released in the form of bindary, and can be built into the option rom, and the external entrance is very clear. This is why it is said that UEFI Driver expands the entire firmware and improves the overall scalability of the firmware. It may be said that the previous firmware can only run applications, but now it can also load many drivers. Some device drivers written by third parties may not need to be built into the flash, and can be reloaded at the shell stage or loaded at the BDS stage.

As for cross-platform, once a driver is released in binary form, it can run on all platforms that follow the UEFI specification. In this way, everyone can develop in parallel, with platform development and device development carried out separately, speeding up the process and finally coordinating them together . (Its advantages)

Classification of UEFI Drivers



UEFI Drivers can be divided into two categories: UEFI Driver Model Driver and UEFI Non Driver Model Driver.

Non Model Driver :

1. No requirements for accessing hardware
2. No requirements for providing services
3. Just comply with EFI Driver specifications, more casual

Driver Model Driver (with protocol)

1. No direct hardware operation
2. No services provided
3. Support loading and unloading, in compliance with Driver Binding Protocol specifications

1>. Service Drivers

This type of Driver will load one or more Protocols on one or more Service Handles and return EFI_SUCCESS in its Entry Point.

2>. Initializing Drivers

This type of driver will not generate any handles, nor will it add any protocols to the handle database. It will only perform some initialization operations and return an error code. Therefore, this type of driver will be unloaded from the system memory after execution.

3>. Root Bridge Drivers

This type of driver will generate one or more control handles, and include a device path protocol and a protocol that abstracts the I/O resources provided by the chip and bus in a software manner. The most common is a driver that generates handles for the PCI Root Bridge, and the generated handle supports the Device Path Protocol and the PCI Root Bridge I/O Protocol.

4>. UEFI Driver Model Drivers

a. Bus Drivers

This type of driver will generate one or more driver handles or driver image handles in the handle database, and install one or more instances of the driver binding protocol on the handle. This type of driver will generate new child handles when calling the start() function of the driver binding protocol, and will add another I/O protocol to the new child handles.

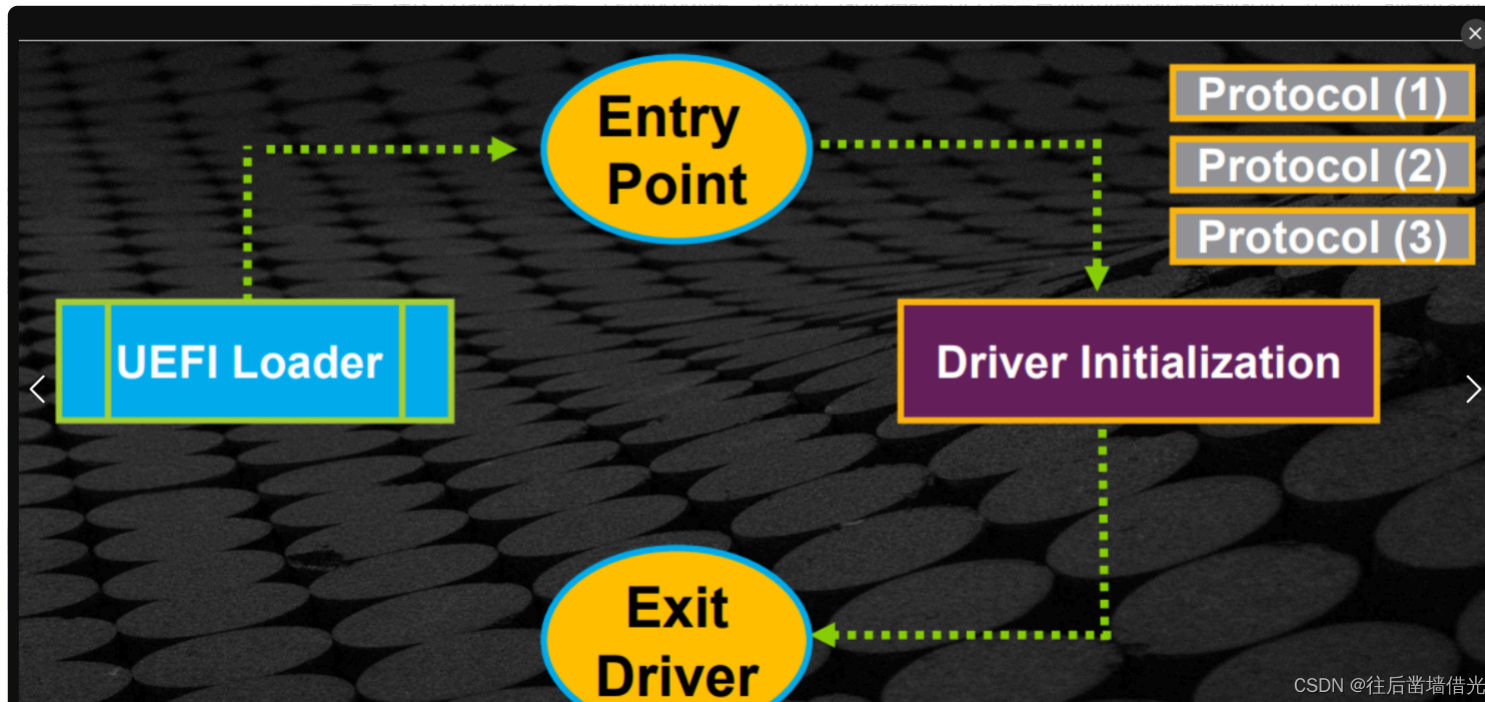
b. Device Drivers

The difference from Bus Drivers is that no new Child Handles will be generated, only additional I/O Protocols will be added to the existing Child Handles.

c. Hybrid Drivers

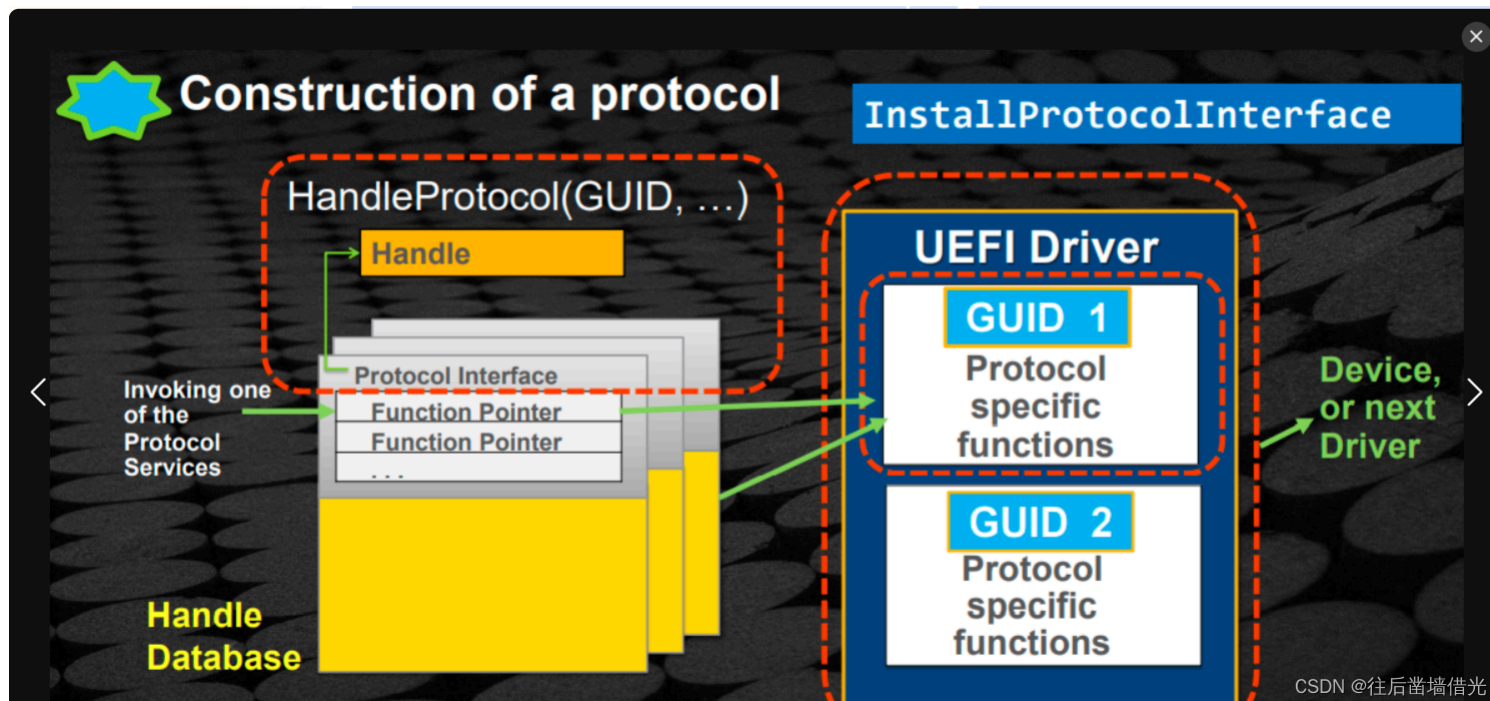
It has the characteristics of both Bus Drivers and Device Drivers, which not only adds I/O Protocol to the existing Handle, but also generates new Child Handles.

Definition of UEFI Driver



When loading in the DXE phase, the UEFI entry point will be executed. After execution, the driver will be initialized and the driver's protocol will be installed. If it is a UEFI Driver, the Driver binding Protocol and component name Protocol will be installed. In this way, even if all the Driver initializations are executed, the entry point will exit and return control to the UEFI Loader.

The application is finished at the above stage, and no trace exists, but it is not finished for the UEFI Driver. Its protocols have been installed and are in the Handle database. The entire UEFI system has begun to manage it. Although the entry point has been executed, the protocol is still there and can still be used by other modules. Therefore, the driver is always resident in the memory until the control is transferred from the BIOS to the OS Loader through the exitbootservice, and then it will be completely released. Or the driver of a module is manually uninstalled during its existence.



Handle Database is used for Protocol management. Suppose a UEFI Driver has two protocols with different names and provides different services, then the protocol will be installed in the entrypoint and installed in the handle database. This protocol may also depend on other installed protocols. It is not necessarily that this protocol reflects a complete implementation from beginning to end. The reason why the BIOS startup system is modularized is to complete the startup through the coordinated loading between modules. Almost all drivers in the DXE stage will call the protocol. The above figure shows that when the UEFI Driver is loaded, it depends on the loaded Protocol in the handle database, and then it will be called by other protocols after successful installation to finally manage the Device.

UEFI Driver Binding Protocol

UEFI Driver Binding Protocol is part of the UEFI specification. It is used to trigger the `Start()` interface to bind the corresponding IO Protocol to the Device Handle after matching the Device Handle in the UEFI framework.

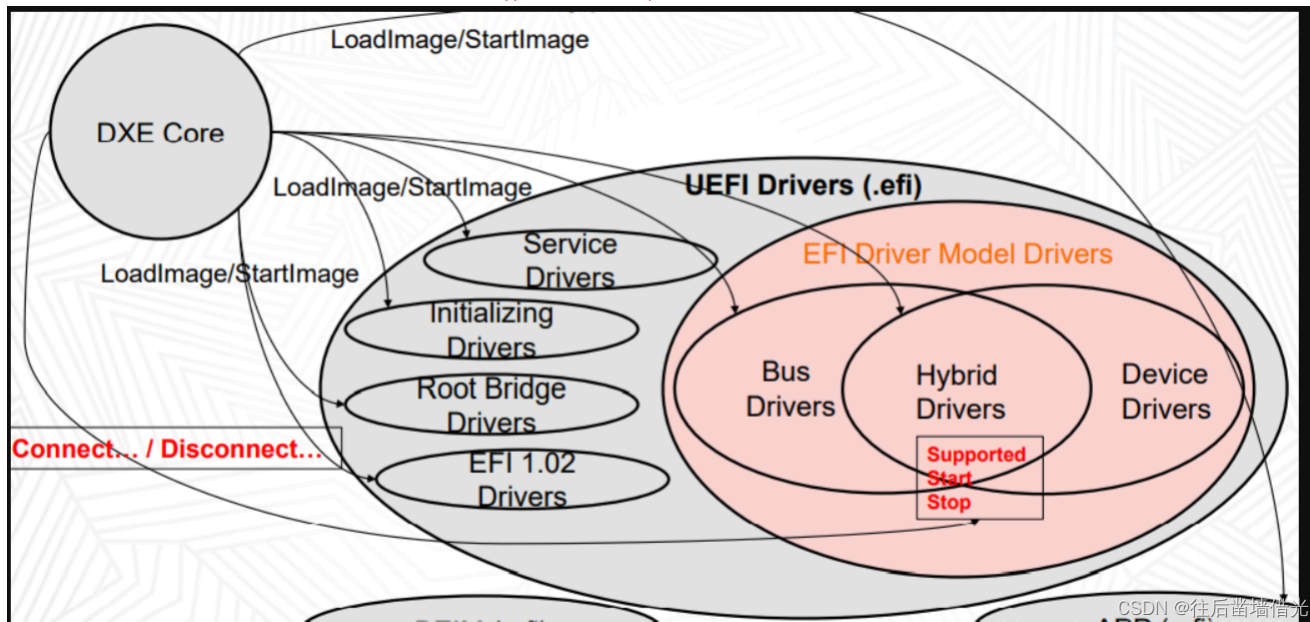
Install Driver Binding Protocol is a protocol defined by the UEFI spec, which contains **three APIs: Supported(), Start(), and Stop()**.

Supported(): Check DeviceID and Vendor ID to determine whether a device is managed by this Driver. If so, UEFI's `Start()` will be started. Polling is executed many times, which will affect resource usage and startup time, so there is no other arrangement except checking.

Start(): It is started after being confirmed by `Supported()`, installs some sub-handles on the hardware controller, and then installs some provided services (Protocol) and Device path on the handle for use by others.

Stop(): Usually it is not called because it also takes time and space. However, from the perspective of UEFI architecture, in some cases, when the device is no longer needed, the driver needs to be released. After calling `Stop()`, all installed protocols will be uninstalled, and the device will no longer be managed and the allocated resources will be freed. However, the driver and the installed driver binding protocol are still there and can be managed and used again next time.

提示：BDS阶段有时会通过connect/Disconnect service来调用设备 (Supported/Start/Stop)，虽然设备再DXE阶段已经存在或已Load准备好了



Take the ScsiDisk driver as an example. When the ScsiDisk driver is loaded, it will first install a **BLOCK_IO_PROTOCOL** and another **SCSI_DRIVER_BINDING_PROTOCOL**. When the system matches the ScsiDisk driver in the UEFI framework, it will call the Support function of the **SCSI_DRIVER_BINDING_PROTOCOL**. If the driver meets the requirements, it will execute the Start function and bind the driver to the BlockIO protocol of the device, so that the BlockIOProtocol interface of the device can be used in applications or services to read and write the device.

DiskIoDxe Driver Example

Link: [UEFI_DRIVER](#)