EDK II source code analysis --- USB protocol EHCI (example) 1

原创 Anthony Posted on 2022-01-05 14:33:25 Read 3k Collection 11 Likes

Category Column: UEFI Article Tags: UEFI

摘要 This article details the different interface standards of USB controllers, including OHCI, UHCI, EHCI, and XHCI, as well as their roles in USB device management. The focus is on the USB host controller driver of EHCI (Enhanced Host Controller Interface), especially the logic of the `EhcDriverBindingStart` function, and the role of URB (USB Request Block) in data transmission. In addition, the management and transmission process of USB device drivers are mentioned, including the processing …

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

Base

1. OHCI (Open Host Controller Interface) is a standard that supports USB1.1, but it is not only for USB, but also supports some other interfaces, such as Apple's Firewire (IEEE 1394) interface. Compared with UHCI, OHCI's hardware is complex and the hardware does more things, so it is relatively simple to implement the corresponding software driver task. It is mainly used for non-x86 USB, such as expansion cards and USB host controllers of embedded development boards.
2. UHCI (Universal Host Controller Interface) is an interface standard for USB1.0 and 1.1 led by Intel and is incompatible with OHCI. UHCI's software driver task is heavy and needs to be more complicated, but it can use USB controllers with cheaper and simpler hardware. Intel and VIA use UHCI, while other hardware providers use OHCI.
3. EHCI (Enhanced Host Controller Interface) is an interface standard for USB2.0 led by Intel. EHCI only provides high-speed functions of USB2.0, and relies on UHCI or OHCI to provide support for full-speed or low-speed devices.
4. XHCI (eXtensible Host Controller Interface) is the latest USB3.0 interface standard. It has greatly improved the speed, energy saving, virtualization and other aspects compared with the previous three. xHCI supports USB devices of all speeds (USB 3.0 SuperSpeed, USB 2.0 Low-, Full-, and High-speed, USB 1.1 Low- and Full-speed). The purpose of xHCI is to replace the previous three (UHCI/OHCI/EHCI).

The USB protocol stack in EDKII consists of three driver programs:

USB host controller driver, USB bus driver and USB device driver

in:

The USB host controller driver source code is located in the MdeModulePkg\Bus\Pci directory

The USB bus driver and USB device driver source code are located in the MdeModulePkg\Bus\Usb directory.

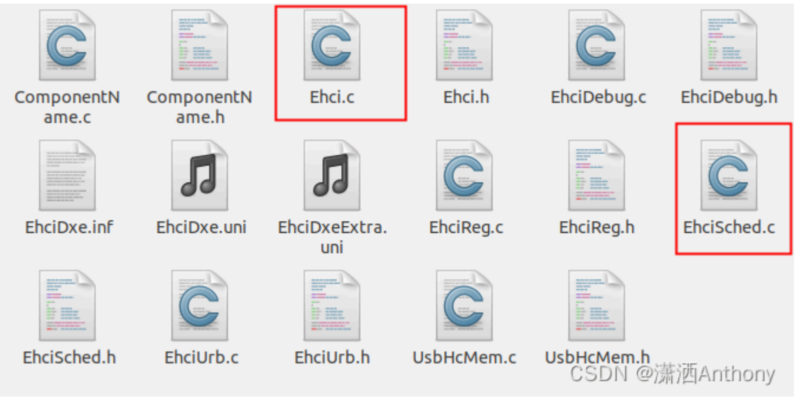
USB host controller driver (HCDI: **EFI_USB2_HC_PROTOCOL**)

USB bus driver (USBDI: **EFI_USB_IO_PROTOCOL**)

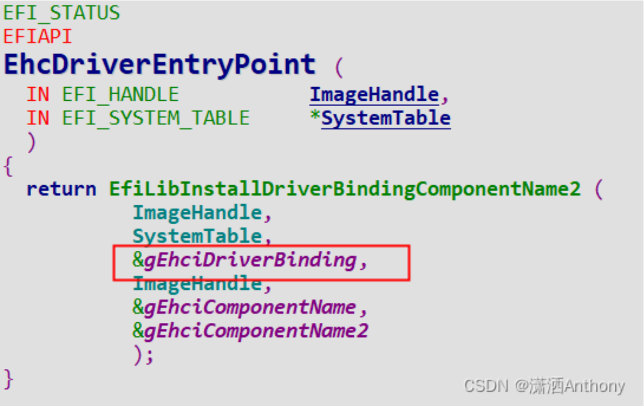
USB device drivers

Take EHCI as an example:

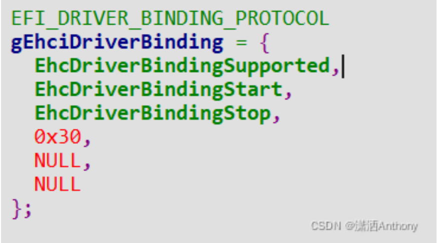
Let's start with the USB host controller driver. The code mainly focuses on the EhciDxe driver. What kind of driver is this? A driver that conforms to the UEFI driver model.



In Ehci.c:



```
EFI_STATUS
EFIAPI
EhcDriverEntryPoint (
  IN EFI_HANDLE           ImageHandle,
  IN EFI_SYSTEM_TABLE     *SystemTable
  )
{
  return EfiLibInstallDriverBindingComponentName2 (
           ImageHandle,
           SystemTable,
           &gEhciDriverBinding,
           ImageHandle,
           &gEhciComponentName,
           &gEhciComponentName2
           );
}
```

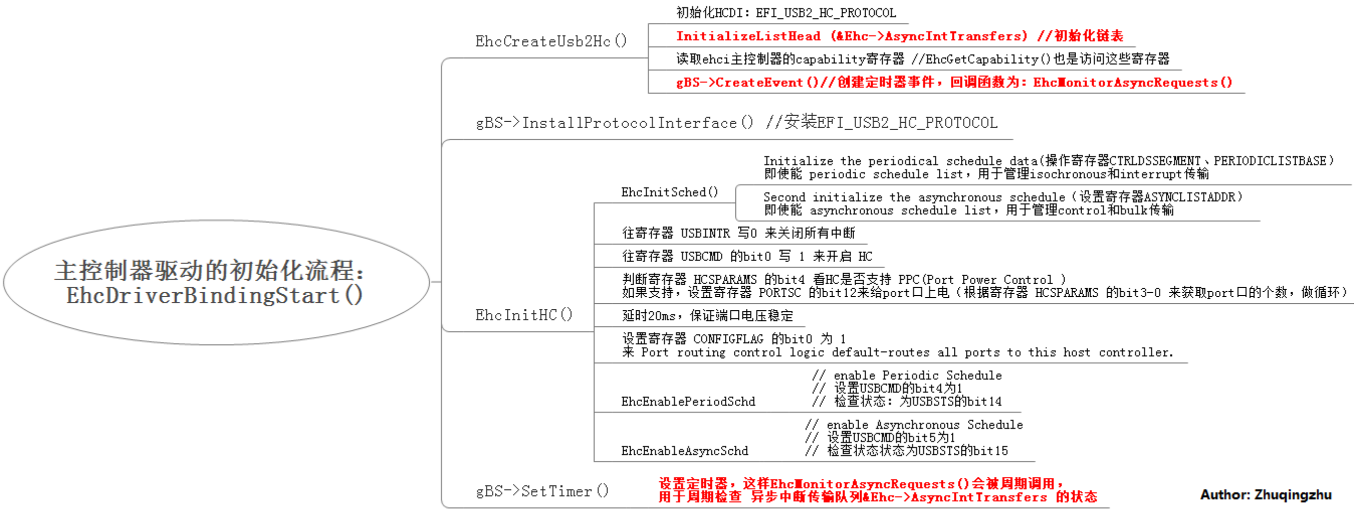


```
EFI_DRIVER_BINDING_PROTOCOL
gEhciDriverBinding = {
  EhcDriverBindingSupported,
  EhcDriverBindingStart,
  EhcDriverBindingStop,
  0x30,
  NULL,
  NULL
};
```

Mainly look at the following function:

```
EFI_STATUS
EFIAPI
EhcDriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *This,
  IN EFI_HANDLE                  Controller,
  IN EFI_DEVICE_PATH_PROTOCOL    *RemainingDevicePath
  )
{。。。。。
```

The function logic is as follows:



主控制器驱动的初始化流程：
EhcDriverBindingStart()

EhcCreateUsb2Hc()
初始化HCDI：EFI_USB2_HC_PROTOCOL
InitializeListHead (&Ehc->AsyncIntTransfers) //初始化链表
读取ehci主控制器的capability寄存器 //EhcGetCapability()也是访问这些寄存器
gBS->CreateEvent()//创建定时器事件，回调函数为：EhcMonitorAsyncRequests()

gBS->InstallProtocolInterface() //安装EFI_USB2_HC_PROTOCOL

EhcInitHC()
EhcInitSched()
Initialize the periodical schedule data(操作寄存器CTRLDSSEGMENT、PERIODICLISTBASE)
即使能 periodic schedule list，用于管理isochronous和interrupt传输
Second initialize the asynchronous schedule（设置寄存器ASYNCLISTADDR）
即使能 asynchronous schedule list，用于管理control和bulk传输
往寄存器 USBINTR 写0 来关闭所有中断
往寄存器 USBCMD 的bit0 写 1 来开启 HC
判断寄存器 HCSPARAMS 的bit4 看HC是否支持 PPC(Port Power Control)
如果支持，设置寄存器 PORTSC 的bit12来给port口上电（根据寄存器 HCSPARAMS 的bit3-0 来获取port口的个数，做循环）
延时20ms，保证端口电压稳定
设置寄存器 CONFIGFLAG 的bit0 为 1
来 Port routing control logic default-routes all ports to this host controller.
EhcEnablePeriodSchd
// enable Periodic Schedule
// 设置USBCMD的bit4为1
// 检查状态：为USBSTS的bit14
EhcEnableAsyncSchd
// enable Asynchronous Schedule
// 设置USBCMD的bit5为1
// 检查状态状态为USBSTS的bit15

gBS->SetTimer()
设置定时器，这样EhcMonitorAsyncRequests()会被周期调用，
用于周期检查 异步中断传输队列&Ehc->AsyncIntTransfers 的状态

Author: Zhuqingzhu

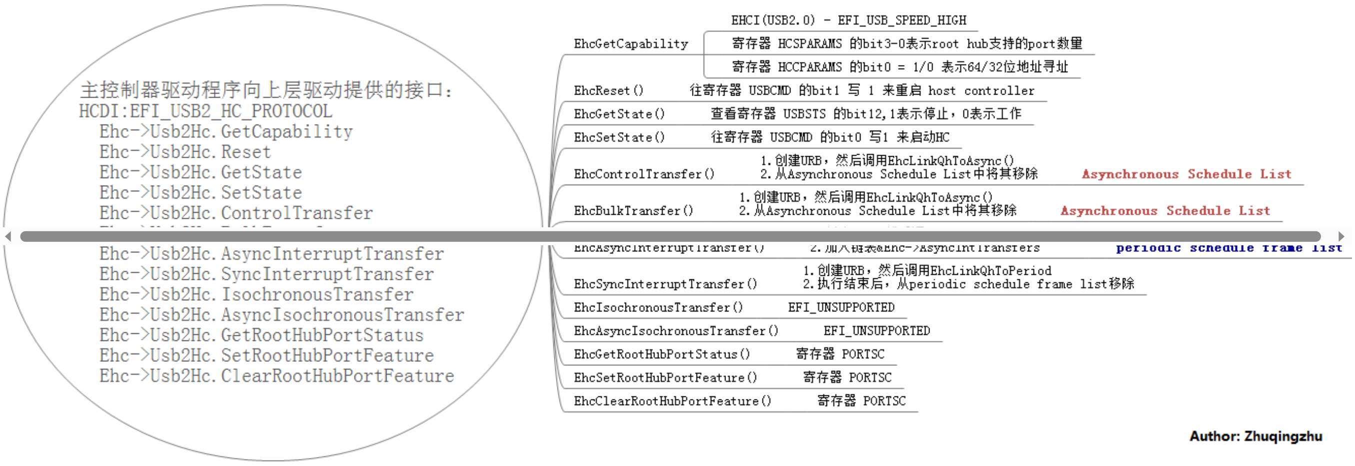
EhcDriverBindingStart starts-----Open PciIo protocol, enable USB host controller-------Open device path protocol on USB host controller----Save original PCI properties----Get Pci device class code-----Determine whether the device is UHCI or OHCI host controller. If so, find out the matching USB ehci host controller and force the ehci driver to connect to it before the UHCI or OHCI driver connects to the UHCI or OHCI host controller------Whether the matching USB host controller is judged, if it is passed, start instantiating USB2_HC_DEV and install EFI_USB2_HC_PROTOCOL, and then the sequence in the above figure is as follows

```
//
  // Init EFI_USB2_HC_PROTOCOL interface and private data structure
  //
  Ehc->Signature                        = USB2_HC_DEV_SIGNATURE;

  Ehc->Usb2Hc.GetCapability             = EhcGetCapability;
  Ehc->Usb2Hc.Reset                     = EhcReset;
  Ehc->Usb2Hc.GetState                  = EhcGetState;
  Ehc->Usb2Hc.SetState                    = EhcSetState;
  Ehc->Usb2Hc.ControlTransfer             = EhcControlTransfer;
  Ehc->Usb2Hc.BulkTransfer                = EhcBulkTransfer;
  Ehc->Usb2Hc.AsyncInterruptTransfer      = EhcAsyncInterruptTransfer;
  Ehc->Usb2Hc.SyncInterruptTransfer       = EhcSyncInterruptTransfer;
  Ehc->Usb2Hc.IsochronousTransfer         = EhcIsochronousTransfer;
  Ehc->Usb2Hc.AsyncIsochronousTransfer    = EhcAsyncIsochronousTransfer;
  Ehc->Usb2Hc.GetRootHubPortStatus        = EhcGetRootHubPortStatus;
  Ehc->Usb2Hc.SetRootHubPortFeature       = EhcSetRootHubPortFeature;
  Ehc->Usb2Hc.ClearRootHubPortFeature     = EhcClearRootHubPortFeature;
  Ehc->Usb2Hc.MajorRevision               = 0x2;
  Ehc->Usb2Hc.MinorRevision               = 0x0;

  Ehc->PciIo                  = PciIo;
  Ehc->DevicePath             = DevicePath;
  Ehc->OriginalPciAttributes = OriginalPciAttributes;
```

收起 ^


主控制器驱动程序向上层驱动提供的接口:
HCDI:EFI_USB2_HC_PROTOCOL
Ehc->Usb2Hc.GetCapability
Ehc->Usb2Hc.Reset
Ehc->Usb2Hc.GetState
Ehc->Usb2Hc.SetState
Ehc->Usb2Hc.ControlTransfer
Ehc->Usb2Hc.AsyncInterruptTransfer
Ehc->Usb2Hc.SyncInterruptTransfer
Ehc->Usb2Hc.IsochronousTransfer
Ehc->Usb2Hc.AsyncIsochronousTransfer
Ehc->Usb2Hc.GetRootHubPortStatus
Ehc->Usb2Hc.SetRootHubPortFeature
Ehc->Usb2Hc.ClearRootHubPortFeature
EhcGetCapability
EHCI(USB2.0) - EFI_USB_SPEED_HIGH
寄存器 HCSPARAMS 的bit3-0表示root hub支持的port数量
寄存器 HCCPARAMS 的bit0 = 1/0 表示64/32位地址寻址
EhcReset()
往寄存器 USBCMD 的bit1 写 1 来重启 host controller
EhcGetState()
查看寄存器 USBSTS 的bit12,1表示停止,0表示工作
EhcSetState()
往寄存器 USBCMD 的bit0 写1 来启动HC
EhcControlTransfer()
1.创建URB,然后调用EhcLinkQhToAsync()
2.从Asynchronous Schedule List中将其移除
Asynchronous Schedule List
EhcBulkTransfer()
1.创建URB,然后调用EhcLinkQhToAsync()
2.从Asynchronous Schedule List中将其移除
Asynchronous Schedule List
EhcAsyncInterruptTransfer()
2.加入链表&Ehc->AsyncIntTransfers
periodic schedule frame list
EhcSyncInterruptTransfer()
1.创建URB,然后调用EhcLinkQhToPeriod
2.执行结束后,从periodic schedule frame list移除
EhcIsochronousTransfer()
EFI_UNSUPPORTED
EhcAsyncIsochronousTransfer()
EFI_UNSUPPORTED
EhcGetRootHubPortStatus()
寄存器 PORTSC
EhcSetRootHubPortFeature()
寄存器 PORTSC
EhcClearRootHubPortFeature()
寄存器 PORTSC

Author: Zhuqingzhu

The above figure only talks about the functions in Ehci.C. In fact, some of the interfaces here eventually need to call the functions in Ehcisched.c. Here we need to understand a concept: what is URB

URB: USB request block, containing information about various data

AI generated projects   登录复制

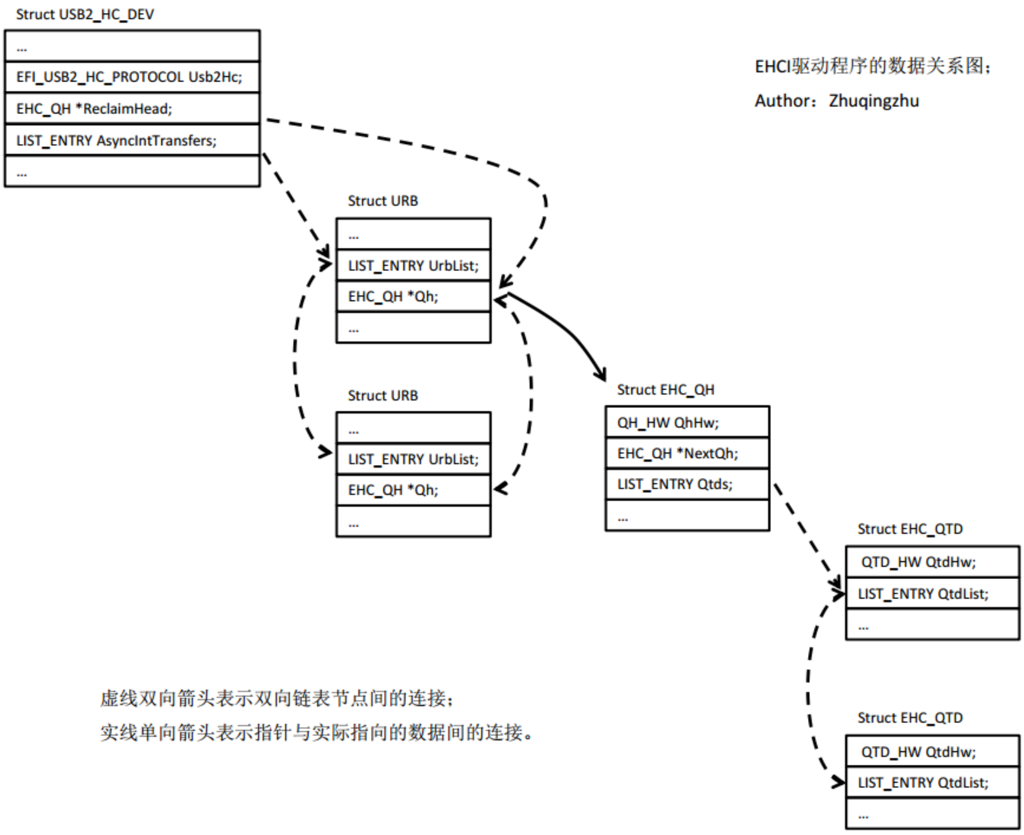
```
struct _URB {
  UINT32                          Signature;
  LIST_ENTRY                      UrbList;

  //
  // Transaction information
  //
  USB_ENDPOINT                    Ep;
  EFI_USB_DEVICE_REQUEST          *Request;     // Control transfer only
  VOID                            *RequestPhy;  // Address of the mapped request
  VOID                            *RequestMap;
  VOID                            *Data;
  UINTN                           DataLen;
  VOID                            *DataPhy;     // Address of the mapped user data
  VOID                            *DataMap;
  EFI_ASYNC_USB_TRANSFER_CALLBACK Callback;
  VOID                            *Context;

  //
  // Schedule data
  //
  EHC_QH                          *Qh;

  //
  // Transaction result
  //
  UINT32                          Result;
  UINTN                           Completed;    // completed data length
  UINT8                           DataToggle;
};
```

收起 ^



1. Struct USB2_HC_DEV is the core data structure of the Host controller, which is created during the initialization process; the data structure definitions of QTD and QH are located in EHCI spec 3.5/3.6;

2. Manage controller and bulk transfers: insert into the Asynchronous Schedule list

```
//把组装好的Qh插入EHCI主控制器的Asynchronous Schedule List,以便硬件执行传输命令
  EhcLinkQhToAsync (Ehc, Urb->Qh);
//阻塞式的执行此次controller传输
  Status = EhcExecTransfer (Ehc, Urb, TimeOut);
//从Asynchronous Schedule List中将其移除
  EhcUnlinkQhFromAsync (Ehc, Urb->Qh);
```

3. Manage isochronous and interrupt transmissions: insert the Periodic schedule frame list

```
//把组装好的Qh插入EHCI主控制器的Periodic schedule frame list，以便硬件执行传输命令
  EhcLinkQhToPeriod (Ehc, Urb->Qh);
//并把URB插入异步中断传输链表 &Ehc->AsyncIntTransfers
  InsertHeadList (&Ehc->AsyncIntTransfers, &Urb->UrbList);
```

4. Insert the URB in the hardware linked list, and the hardware will automatically execute the send;

5. The linked list &Ehc->AsyncIntTransfers is created and managed by the driver and managed by **EhcMonitorAsyncRequests()** ;

(1) He will loop each urb on &Ehc->AsyncIntTransfers;

(2) Determine the execution result by judging QTD.status (a urb contains a QH and a string of QTDs);

(3) Update QH to prepare for the next round of asynchronous transmission;

(4) If there is a callback function, execute the callback function.

Let me emphasize: Execute the controller transfer in a blocking manner: **Status = EhcExecTransfer (Ehc, Urb, TimeOut);**

```
EFI_STATUS
EhcExecTransfer (
  IN  USB2_HC_DEV         *Ehc,
  IN  URB                 *Urb,
  IN  UINTN               TimeOut
  )
{
  EFI_STATUS                Status;
  UINTN                     Index;
  UINTN                     Loop;
  BOOLEAN                   Finished;
  BOOLEAN                   InfiniteLoop;

  Status       = EFI_SUCCESS;
  Loop         = TimeOut * EHC_1_MILLISECOND;
  Finished     = FALSE;
  InfiniteLoop = FALSE;

  //
  // According to UEFI spec section 16.2.4, If Timeout is 0, then the caller
  // must wait for the function to be completed until EFI_SUCCESS or EFI_DEVICE_ERROR
  // is returned.
  //
  if (TimeOut == 0) {
    InfiniteLoop = TRUE;
  }

  for (Index = 0; InfiniteLoop || (Index < Loop); Index++) {
    Finished = EhcCheckUrbResult (Ehc, Urb);

    if (Finished) {
      break;
    }

    gBS->Stall (EHC_1_MICROSECOND);
  }

  if (!Finished) {
    DEBUG ((EFI_D_ERROR, "EhcExecTransfer: transfer not finished in %dms\n", (UINT32)TimeOut));
    EhcDumpQh (Urb->Qh, NULL, FALSE);

    Status = EFI_TIMEOUT;

  } else if (Urb->Result != EFI_USB_NOERROR) {
    DEBUG ((EFI_D_ERROR, "EhcExecTransfer: transfer failed with %x\n", Urb->Result));
    EhcDumpQh (Urb->Qh, NULL, FALSE);

    Status = EFI_DEVICE_ERROR;
  }

  return Status;
}
```

收起 ^

What if the device fails in this function? The phenomenon is that it will keep retrying until the maximum number of retries is reached. If the device still fails, it will be skipped directly. However, there is a problem. It usually takes several minutes to reach the maximum number of retries. This can easily create the illusion that you cannot enter the system and have to wait until the retry is completed. This situation usually occurs in industrial computers. Due to the large number of external USB devices, USB device abnormalities cannot be ruled out.

OK, let's analyze USB device in the next section.