JEFI Basic Tutorial (XIX) - Simple Use of PCIe

原创 xiaopangzi313 Posted on 2023-01-01 19:58:52 Read 2.1k Collection 4 Likes 2

Category Column: 15_Firmware Development Article Tags: Development Language C language

摘要 This article shows how to locate the PCI root bridge and enumerate PCI devices in UEFI applications. Through the `LocatePciRootBridgeIo` and `PciDevicePresent` functions, the program looks for `gPciRootBridgeIoProtocol`, reads the device information, and uses `PciIoProtocol` to access the device. The code implements the PCI device search process starting from the EFI entry point.

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

.. Write source code

AI generated projects 登录复制

```
//OvmfPkg/HelloWorldPci/HelloWorldPci.c
#include <Uefi.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/ShellCEntryLib.h>
#include <Library/DebugLib.h>

#include <Protocol/PciIo.h>
#include <Protocol/PciRootBridgeIo.h>
#include <IndustryStandard/Pci.h>


EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL *gPciRootBridgeIo;

EFI_STATUS LocatePciRootBridgeIo(void);

EFI_STATUS PciDevicePresent(
  IN EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL * PciRootBridgeIo,
  OUT PCI_TYPE00 * Pci,
  IN UINT8 Bus,
  IN UINT8 Device,
  IN UINT8 Func
  );

EFI_STATUS ListPciInformation(void);


EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
  EFI_STATUS Status;


  Status = LocatePciRootBridgeIo();
  if(EFI_ERROR(Status))
  {
    DEBUG((DEBUG_ERROR, "[CSDN]Call LocatePciRootBridgeIo failed,Can't find protocol!\n"));
  }
```

```c
  else
  {
    DEBUG((DEBUG_ERROR, "[CSDN]Call LocatePciRootBridgeIo successed,Find protocol!\n"));
  }


  ListPciInformation();

  return EFI_SUCCESS;
}

EFI_STATUS LocatePciRootBridgeIo()
{
  EFI_STATUS Status;
  EFI_HANDLE *PciHandleBuffer = NULL;
  UINTN      HandleIndex = 0;
  UINTN      HandleCount = 0;

  Status = gBS->LocateHandleBuffer(
    ByProtocol,
    &gEfiPciRootBridgeIoProtocolGuid,
    NULL,
    &HandleCount,
    &PciHandleBuffer
    );
  if(EFI_ERROR(Status))  return Status;

  for(HandleIndex = 0; HandleIndex < HandleCount; HandleIndex++)
  {
    Status = gBS->HandleProtocol(
      PciHandleBuffer[HandleIndex],
      &gEfiPciRootBridgeIoProtocolGuid,
      (VOID **)&gPciRootBridgeIo
      );
    if(EFI_ERROR(Status))  continue;
    else                   return EFI_SUCCESS;
  }

  return Status;

}

EFI_STATUS ListPciInformation()
{
  EFI_STATUS Status = EFI_SUCCESS;
  PCI_TYPE00 Pci;
  UINT16 Dev,Func,Bus,PciDevicecount = 0;

  DEBUG((DEBUG_ERROR, "[CSDN] PciDeviceNo\tBus\tDev\tFunc | Vendor.Device.ClassCode\n"));
  for(Bus=0; Bus<256; Bus++) {
    for(Dev=0; Dev<= PCI_MAX_DEVICE; Dev++)
      for(Func=0; Func<=PCI_MAX_FUNC; Func++)
      {
        Status = PciDevicePresent(gPciRootBridgeIo,&Pci,(UINT8)Bus,(UINT8)Dev,(UINT8)Func);
          if(Status == EFI_SUCCESS)
          {
            PciDevicecount++;
            DEBUG((DEBUG_ERROR, "[CSDN] %d\t\t%x\t%x\t%x\t",PciDevicecount,(UINT8)Bus,(UINT8)Dev,(UINT8)Func));
            DEBUG((DEBUG_ERROR, "%x\t%x\t%x\n",Pci.Hdr.VendorId,Pci.Hdr.DeviceId,Pci.Hdr.ClassCode[0]));
          }
      }
  }
  return EFI_SUCCESS;
}
```

```
EFI_STATUS
PciDevicePresent (
  IN  EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL     *PciRootBridgeIo,
  OUT PCI_TYPE00                          *Pci,
  IN  UINT8                               Bus,
  IN  UINT8                               Device,
  IN  UINT8                               Func
  )
{
  UINT64      Address;
  EFI_STATUS  Status;

  //
  // Create PCI address map in terms of Bus, Device and Func
  //
  Address = EFI_PCI_ADDRESS (Bus, Device, Func, 0);

  //
  // Read the Vendor ID register
  //
  Status = PciRootBridgeIo->Pci.Read (
                                  PciRootBridgeIo,
                                  EfiPciWidthUint32,
                                  Address,
                                  1,
                                  Pci
                                  );

  if (!EFI_ERROR (Status) && (Pci->Hdr).VendorId != 0xffff) {
    //
    // Read the entire config header for the device
    //
    Status = PciRootBridgeIo->Pci.Read (
                                    PciRootBridgeIo,
                                    EfiPciWidthUint32,
                                    Address,
                                    sizeof (PCI_TYPE00) / sizeof (UINT32),
                                    Pci
                                    );

    return EFI_SUCCESS;
  }

  return EFI_NOT_FOUND;
}
```

收起 ^

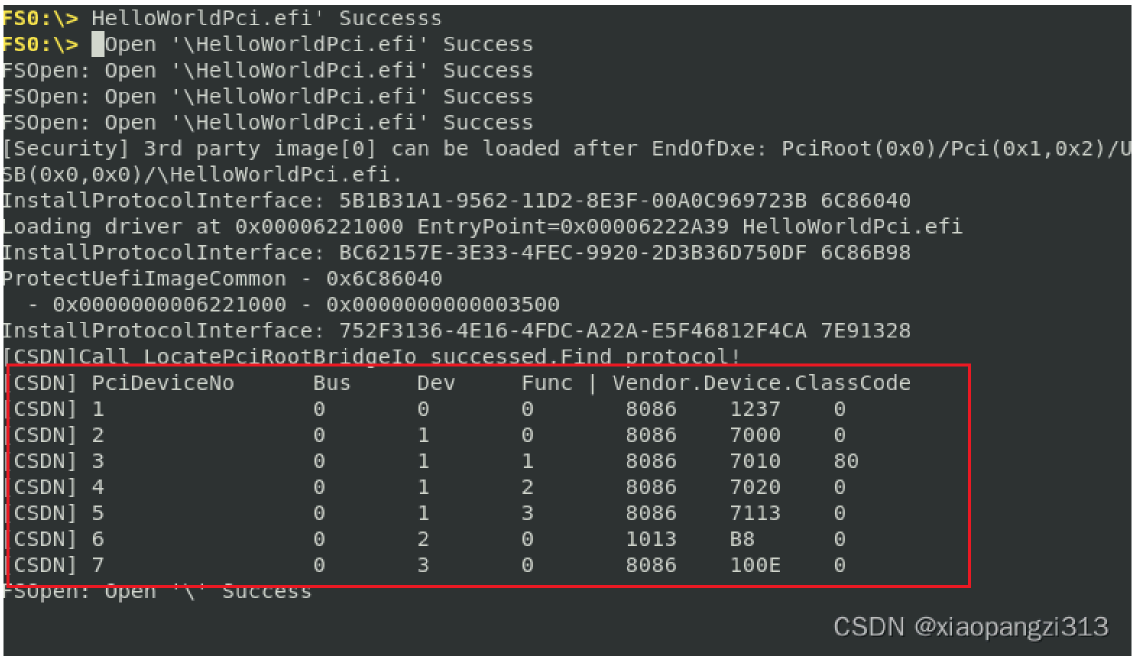
AI generated projects 登录复制

```
//OvmfPkg/HelloWorldPci/HelloWorldPci.inf
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = HelloWorldPci
  FILE_GUID                      = 6987935E-ED34-44db-AE97-1FA5E4ED2116
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 1.0
  ENTRY_POINT                    = UefiMain

#
#  This flag specifies whether HII resource section is generated into PE image.
#
  UEFI_HII_RESOURCE_SECTION      = TRUE

#
# The following information is for reference only and not required by the build tools.
```

```
#
#  VALID_ARCHITECTURES           = IA32 X64 IPF EBC
#

[Sources]
  HelloWorldPci.c


[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec
  ShellPkg/ShellPkg.dec

[LibraryClasses]
  UefiApplicationEntryPoint
  UefiShellCEntryLib
  BaseLib
  BaseMemoryLib
  DebugLib
  UefiBootServicesTableLib
  MemoryAllocationLib
  UefiLib
  UefiLib
  PcdLib
```

收起 ^

. Compile and generate EFI files & run



```
FS0:\> HelloWorldPci.efi' Successs
FS0:\> Open '\HelloWorldPci.efi' Success
FSOpen: Open '\HelloWorldPci.efi' Success
FSOpen: Open '\HelloWorldPci.efi' Success
FSOpen: Open '\HelloWorldPci.efi' Success
[Security] 3rd party image[0] can be loaded after EndOfDxe: PciRoot(0x0)/Pci(0x1,0x2)/U
SB(0x0,0x0)/\HelloWorldPci.efi.
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 6C86040
Loading driver at 0x00006221000 EntryPoint=0x00006222A39 HelloWorldPci.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 6C86B98
ProtectUefiImageCommon - 0x6C86040
  - 0x0000000006221000 - 0x0000000000003500
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7E91328
[CSDN]Call LocatePciRootBridgeIo successed.Find protocol!
[CSDN] PciDeviceNo      Bus      Dev      Func | Vendor.Device.ClassCode
[CSDN] 1                0        0        0       8086     1237     0
[CSDN] 2                0        1        0       8086     7000     0
[CSDN] 3                0        1        1       8086     7010     80
[CSDN] 4                0        1        2       8086     7020     0
[CSDN] 5                0        1        3       8086     7113     0
[CSDN] 6                0        2        0       1013     B8       0
[CSDN] 7                0        3        0       8086     100E     0
FSOpen: Open '\' Success
```

. Summary

EFI The enumeration process in `PCIe device` mainly creates two `Protocol` , namely `gPciRootBridgeIoProtocol` and `PciIoProtocol` . mong them, gPciRootBridgeIoProtocol corresponds to a RootBridge (RootCompex), which implements `RootBridge` all `PCIe` read and write nplementations under . `PciIoProtocol` Corresponding to each enumerated device. If you want to `UEFI` access the device under `PCIe` , you an use these two `Protocol` .