

PCI/PCIE related knowledge

原创

Anthony

Posted on 2022-01-12 15:01:31

Read 1.5k

Collection 4

Likes

Category Column: BIOS Basics


Article Tags:

UEFI

PCI-E

pci

Copyright CC 4.0 BY-SA

 BIOS Basics

This column includes this content

6 articles

Subscribe to our column

摘要

This article introduces the basics of PCI/PCIE configuration space, explains in detail the two access methods of configuration space: IO mode and MMIO mode, and shows how to read key information in configuration registers through code.

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

PCI /e Configuration Space

The PCI/ **PCIE** Configuration Space is an independent space that is parallel to the Memory Space and IO Space. The PCI Configuration Space has 256 Bytes, and the PCIE Configuration Space is expanded to 4096 Bytes. PCIE is a protocol developed on the basis of PCI, and it is expanded on this basis. Its expansion is completed through a register block called Capability.

31		16		15		0		
Device ID				Vendor ID				00h
Status				Command				04h
Class Code						Revision ID		08h
BIST		Header Type		Latency Timer		Cachelline Size		0Ch
Base Address Registers								10h
								14h
								18h
								1Ch
								20h
								24h
Cardbus CIS Pointer								28h
Subsystem ID				Subsystem Vendor ID				2Ch
Expansion ROM Base Address								30h
Reserved						Capabilities Pointer		34h
Reserved								38h
Max_Lat		Min_Gnt		Interrupt Pin		Interrupt Line		3Ch

CSDN @清酒Anthony

Access method

IO Mode

The CPU provides two sets of I/O registers for accessing the configuration space:

- Configuration Space Control Registers CF8h-CFBh
- Configuration space data registers CFCh-CFFh

Traditionally, writing to IO ports CFCh and CF8h can only access the first 256 bytes of the PCI/PCIE device (the configuration space of the PCI device is only 256 bytes...)

```
1 // for PCI
2 address = BIT31 | ((Bus & 0xFF) << 16) | ((Dev & 0x1F) << 11) | ((Fun & 0x7) << 8) | (Reg & 0xfffffff); //BIT31=0x80000000
```

```

3 | 4 | // write cfg register
5 | IoWrite8(0xcfc, address);
6 | // read data register
7 | data8 = IoRead8(0xcfc);
8 | data16 = IoRead16(0xcfc);
9 | data32 = IoRead32(0xcfc);

```

收起 ^

Bit 31 represents the enable bit. It must be set, otherwise it will not work.

MMIO mode

The access to MMIO is the same as the way to access memory. It starts from the base address called PCIEXBAR. There is a large space. The value of this PCIEXBAR may vary depending on the platform. The possible values are 0xB0000000, 0xC0000000, etc. (Feitian uses 0xB0000000).

AI generated projects

登录复制

```

1 | #define pcie_addr(m, b, d, f, o)      (m + ((b & 0xff) << 20) + ((d & 0x1f) << 15) + ((f & 0x7) << 12) + (o & 0xffffffffc))
2 | #define mmio_read8(addr)             (*(volatile uint8 *)addr)
3 | #define mmio_write8(addr, data8)     *(volatile uint8 *)addr = data8
4 | #define mmio_read16(addr)            (*(volatile uint16 *)addr)
5 | #define mmio_write16(addr, data16)   *(volatile uint16 *)addr = data16
6 | #define mmio_read32(addr)            (*(volatile uint32 *)addr)
7 | #define mmio_write32(addr, data32)   *(volatile uint32 *)addr = data32

```

The configuration register group retains readable information that describes the basic characteristics of the PCI device in detail. After the CPU reads this information, it can set the required configuration content for the PCI device, thereby achieving automatic configuration. This readable information includes:

Vendor ID: Equipment vendor number, assigned by the PCI SIG international organization.

Device ID: A specific device number assigned by the device vendor.

Revision ID: A specific revision number for a device, assigned by the device vendor.

Class Code: Functional category number of the device.

Header Type: Indicates the content format of the area from address 10H to 3FH in the Header, and also indicates whether the device is a multi-function device.

When scanning PCI devices, we can directly determine whether the device exists by whether the Vendor ID and Device ID are 0xffff. If so, the device does not exist. However, there is also a situation where the domestic products are not complete. When designing the hardware, it is very likely that the non-existent value is not 0xffff, and it may be 0.

AI generated projects

登录复制

```

1 | //
2 | // Create PCI address map in terms of Bus, Device and Func
3 | //
4 | Address = EFI_PCI_ADDRESS (Bus, Device, Func, 0);
5 |
6 | //
7 | // Read the Vendor ID register
8 | //
9 | Status = PciRootBridgeIo->Pci.Read (
10 |         PciRootBridgeIo,
11 |         EfiPciWidthUint32,
12 |         Address,
13 |         1,
14 |         Pci
15 |         );
16 |
17 | if (!EFI_ERROR (Status) && (Pci->Hdr).VendorId != 0xffff && (Pci->Hdr).VendorId != 0) {
18 |     //
19 |     // Read the entire config header for the device
20 |     //
21 |     Status = PciRootBridgeIo->Pci.Read (
22 |         PciRootBridgeIo,
23 |         EfiPciWidthUint32,
24 |         Address,
25 |         sizeof (PCI_TYPE00) / sizeof (UINT32),
26 |         Pci
27 |         );
28 |
29 |     return EFI_SUCCESS;
30 | }
31 |
32 |

```

收起 ^

This part of the code is defined in the PciEnumeratorSupport.c file