(4 条消息)【14】PCIe 架构下 memory 空间、IO 空间、PCIe 配置空间简介_linjiasen 的博…

1、 4 种空间迷魂阵

PCIe 架构下定义了 4 种地址空间：Memory 空间、IO 空间、配置空间和 message 空间。

我们先看一下 PCIe spec 关于这四种空间的定义：

（1）配置空间 Configuration Space

One of the four address spaces within the PCI Express architecture. Packets with a Configuration Space address are used to configure Functions。

（2）IO 空间 I/O Space

One of the four address spaces of the PCI Express architecture. Identical to the I/O Space defined in the PCI Local Bus Specification

（3）memory 空间 Memory Space

One of the four address spaces of the PCI Express architecture. Identical to the Memory Space defined in PCI 3.0

（4）message 空间 Message Space

One of the four address spaces of the PCI Express

architecture

看完介绍是不是更懵 X 了，如果没有懵 X 说明这篇文章不适合你了，如果懵 X 了就继续。

message 空间其实就是用来 report 带内的 message 和 event 的（比如 MSI 中断、电源管理消息等），其实就是通过带内 message transaction 的方式来代替带外信号，用 message 的好处就是可以省去许多带外信号。

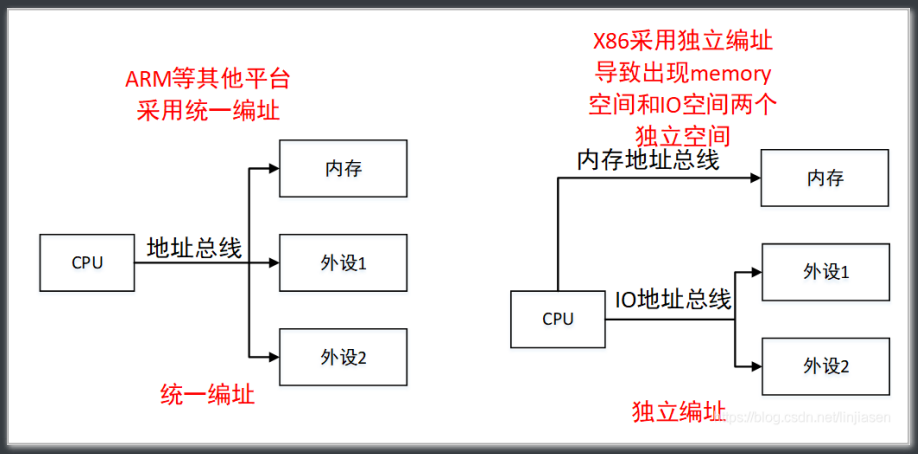
下面是 PCIe spec 中关于 message space 描述的原话：

The transaction Layer supports four address spaces: it includes the three PCI address spaces (memory, I/O, and configuration) and adds Message Space. This specification uses Message Space to support all prior sideband signals, such as interrupts, power-management requests, and so on, as in-band Message transactions. You could think of PCI Express Message transactions as “virtual wires” since their effect is to eliminate the wide array of sideband signals currently used in a platform implementation.

抛开第 4 种 message 空间先不谈，先看其他三种 PCI 架构 就定义的地址空间：Memory 空间、IO 空间、配置空间。说到 memory 空间和 IO 空间就不得不说统一编址和独立编址的问题。

2、 统一编址和独立编址导致 IO 空间和 Memory 空间分

离



X86 采用独立编址的方式，将 memory 操作与外设 IO 操作分开了，才有了 memory 空间和 IO 空间的区分。X86 平台 CPU 内部对 内存 和外设寄存器访问的指令也是不同的。

IO 空间：

访问外部设备寄存器的地址区域，（PCI 支持 4GB 的 IO 空间，但是 x86 平台为 64KB）。

memory 空间：

访问 memory 的地址空间，32 位平台为 4G。 **此 memory 空间和 main memory（平时常说的内存或者主存）是两个概念**，32bit 平台下 CPU memory 地址总线只能寻址到 4G，这 4G 空间包括 main memory、外设 IO 空间映射（MMIO）等，不能全给 main memory，因此

间映射（MMIO）等，不能全部 main memory，因此

32bit 的 CPU 是无法配置 4G 内存的。

PCIe 配置空间：

PCIe spec 规定了所有 PCIe 设备（除了 host bus bridge 外）必须实现配置空间，说白了就是 PCI-SIG 规定了一种独立于 memory 空间的 PCIe 设备访问（读写、配置）机制（说白了就是一堆按规则排列的 reg）。PCI-SIG 详细规定了 PCIe 设备 reg 的排列（每个 capability id reg 的后面 4Byte 会保存 next capability 的起始地址，这样方便芯片厂商扩展 reg，并且 PCIe 驱动软件天然可以使用 list 来管理这些 reg）。

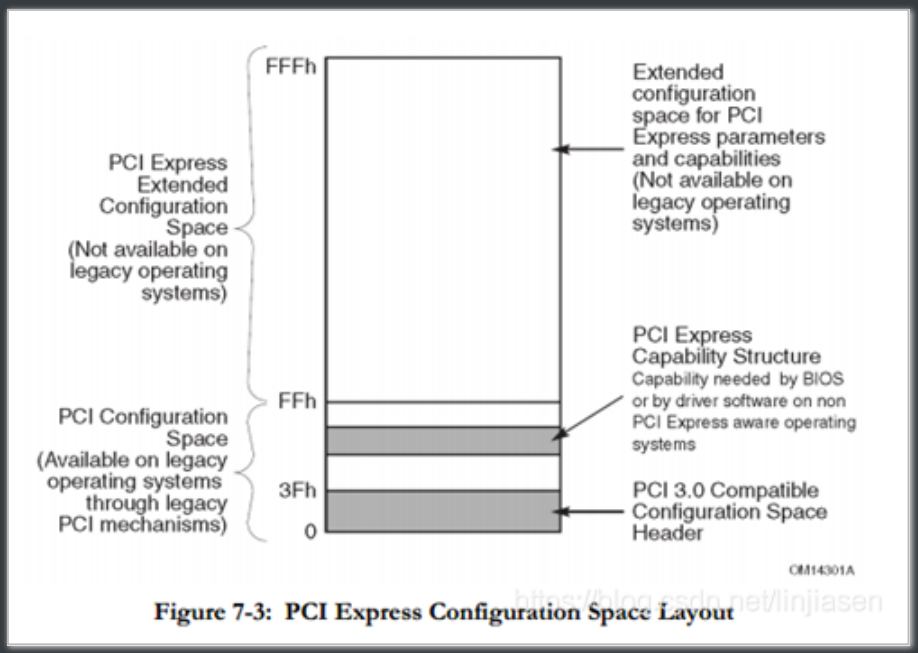


Figure 7-3: PCI Express Configuration Space Layout

3、 三角关系

X86 的 CPU 可以直接访问 memory 空间和 IO 空间，但是不能直接访问 PCIe 配置空间（原因很简单，X86 的 CPU 只有 memory 指令和 IO 指令，没有配置指令）。因此，**需要把 PCIe 配置空间映射到 memory 空间或者 IO 空**

此，**需要把 PCIe 配置空间映射到 memory 空间或者 IO 空间**（一般不推荐映射到 IO 空间）。或者说 CPU 访问 PCIe 配置空间需要一个翻译官（RC）。这个翻译官是干好事的（帮 CPU 的 memory 访问或者 IO 访问转换成 PCIe 域的请求），不是给鬼子带路的汉奸。

在 X86 系统中，IO 方式的翻译官就是 CONFIG_DATA 和 CONFIG_ADDRESS，这个两个位于 IO 空间的端口。这就是所谓的 PCI 的 CAM 方式，只能访问 PCI 兼容的配置空间（前面 256Byte，从下图中也可看到 register number 只有 8bit，所有只能访问 256Byte。**但是有些芯片比较鸡贼，会把 reserved 的 bit24 到 30 用起来，使用 bit24-27 作为 extended register number，和 bit0-7 的 register number 组合这样可以扩展到 4K，不过这种扩展属于芯片的私有行为，并不是所有芯片都支持**）。

在 X86 系统中，Memroy 方式的翻译官就是 MCFG（memory mapped configuration space base address description table 可以通过 **cat /proc/iomem 查看 PCI MMCONFIG 得到在 memory 空间的映射**）其实就是 bus 0 dev 0 function 0 的 BASE 地址。这就是所谓的 PCIe 的 ECAM 方式，可以访问 PCIe 全部 4K 配置空间。
可以通过 cat/proc/iomem | grep MMCONFIG 得到 MMCONFIG 的 base 地址，然后使用 busbox 下面的 devmem 按照表格 7-1 的规则访问配置空间（下面程序找了 0:3.1 和 0:0.0 做了下实验），**但是这种方式存在顺序问题，见 PCIe Spec。**

Table 7-1: Enhanced Configuration Address Mapping

Memory Address[106]	PCI Express Configuration Space
A[(20 + *n* – 1):20]	Bus Number 1 ≤ *n* ≤ 8

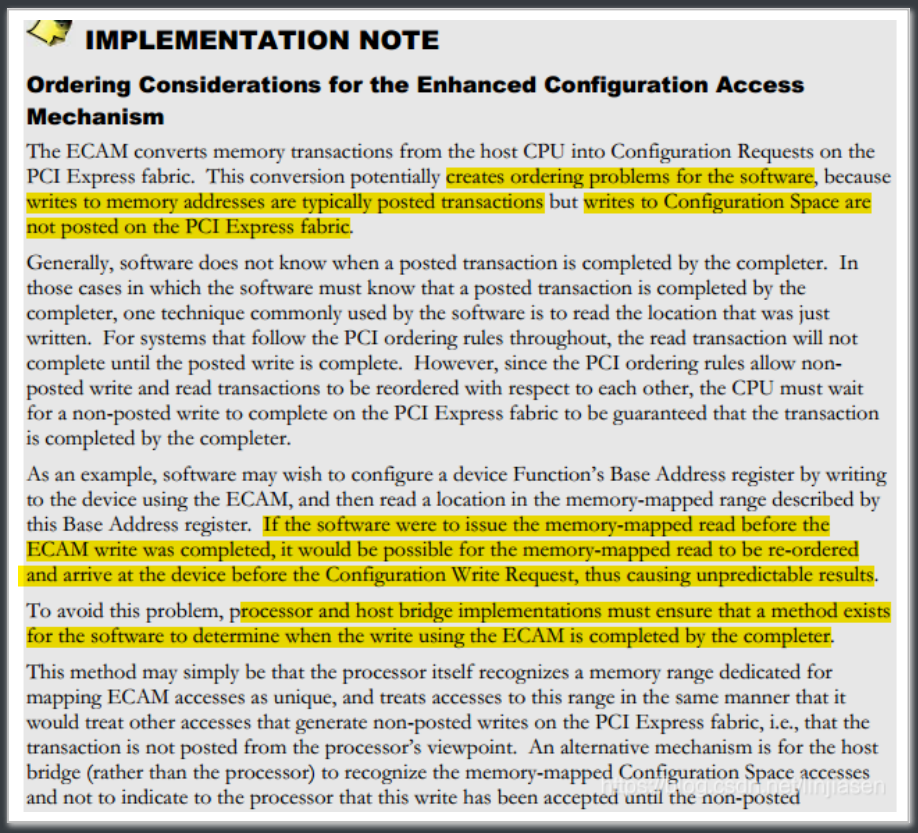
A[19:15]	Device Number
A[14:12]	Function Number
A[11:8]	Extended Register Number
A[7:2]	Register Number
A[1:0]	Along with size of the access, used to generate Byte Enables

ECAM 把 memory 事务从 host CPU 转换成 PCIe fabric 配置请求。这种转换对应软件来说存在潜在的**顺序问题**，因为写 memory 地址是典型的 post 事务（不需要 completion），但是写配置空间是 non post 的请求（需要 completion）。

软件无法知道什么时候，完成者完成了 post 事务。这种场景下（ECAM 访问），软件必须要知道完成者已经完成了 post 请求，**软件通常用回读刚写过 location 的这种方式来确定完成者是否完成**。对于遵守 PCI order 规则的系统，read 事务必须要在 post 写完成后才能完成。然而，由于 PCI order 规则允许 non post 写事务和 read 事务进行乱序处理，CPU 必须等待 PCIe fabric 上 non-post 写请求完成来确保完成者完成了这个事务（也就说，由于允许乱序 non post 写事务插了 read 的队）。

举个例子，软件期望通过 ECAM 的方式写 device 的 Base address reg，然后读取 memory-map 的区域的 base address reg 的位置。如果软件发出 memory-map 读请求乱序了，在配置写请求达到前就达到，将会引起不可预知的结果。

为了阻止这个问题，处理器和主桥必须确保有一种方式可以让软件确定什么时候使用 ECAM 的写请求被完成者完成。



IMPLEMENTATION NOTE

Ordering Considerations for the Enhanced Configuration Access Mechanism

The ECAM converts memory transactions from the host CPU into Configuration Requests on the PCI Express fabric. This conversion potentially creates ordering problems for the software, because writes to memory addresses are typically posted transactions but writes to Configuration Space are not posted on the PCI Express fabric.

Generally, software does not know when a posted transaction is completed by the completer. In those cases in which the software must know that a posted transaction is completed by the completer, one technique commonly used by the software is to read the location that was just written. For systems that follow the PCI ordering rules throughout, the read transaction will not complete until the posted write is complete. However, since the PCI ordering rules allow non-posted write and read transactions to be reordered with respect to each other, the CPU must wait for a non-posted write to complete on the PCI Express fabric to be guaranteed that the transaction is completed by the completer.

As an example, software may wish to configure a device Function's Base Address register by writing to the device using the ECAM, and then read a location in the memory-mapped range described by this Base Address register. If the software were to issue the memory-mapped read before the ECAM write was completed, it would be possible for the memory-mapped read to be re-ordered and arrive at the device before the Configuration Write Request, thus causing unpredictable results.

To avoid this problem, processor and host bridge implementations must ensure that a method exists for the software to determine when the write using the ECAM is completed by the completer.

This method may simply be that the processor itself recognizes a memory range dedicated for mapping ECAM accesses as unique, and treats accesses to this range in the same manner that it would treat other accesses that generate non-posted writes on the PCI Express fabric, i.e., that the transaction is not posted from the processor's viewpoint. An alternative mechanism is for the host bridge (rather than the processor) to recognize the memory-mapped Configuration Space accesses and not to indicate to the processor that this write has been accepted until the non-posted

下面我们通过 0:3.1 和 0:0.0 两个设备，验证下 config read 和 ECAM 的 read 是否一致。我们可以看到系统中 mmconfig 的地址是 0xf800_0000，0:3.1 按照 table 7-1 算出的 offset 是 0x1_9000，我们访问 0xf801_9000 的地址就是 0:3.1 的配置空间的 0 地址，也就是 devieid 和 vendorid。可以看出 ECAM 方式访问和配置方式访问的值是一致的。

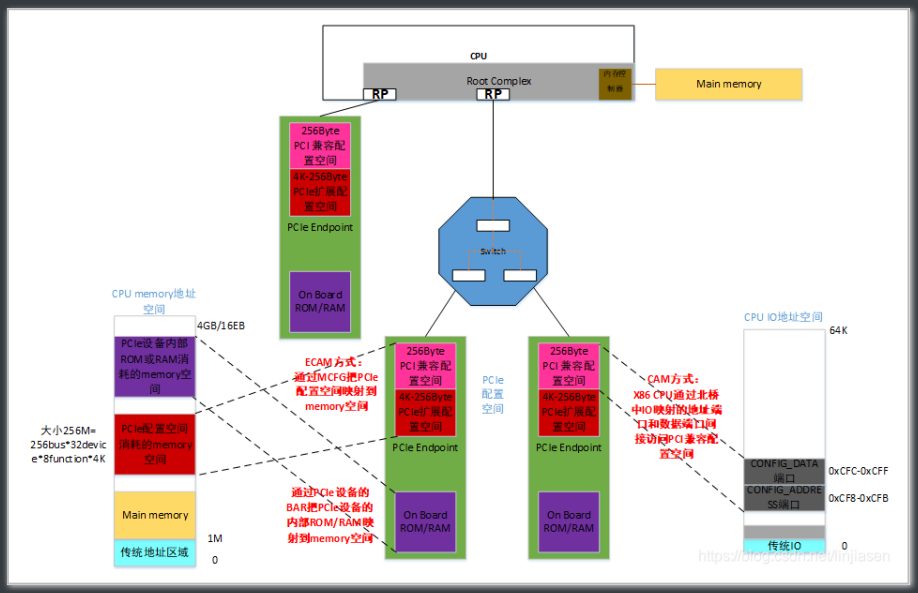
```
铁 busybox git:(master) 参[00m cat bus.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  unsigned int bus = 0;
  unsigned int dev = 3;
  unsigned int function = 1;
  unsigned int mmconfigoffset = 0;
  mmconfigoffset = ((bus<<20) + (dev<<15)+ (function <<12));
  printf("bus 0x%x dev 0x%x function 0x%x memconfig offset 0x%x",
    bus,dev, function, mmconfigoffset);
  return 0;
}

铁 busybox git:(master) 参[00m gcc -o bus bus.c
铁 busybox git:(master) 参./bus
bus 0x0 dev 0x3 function 0x1 memconfig offset 0x19000
铁 busybox git:(master) 参sudo cat /proc/iomem| grep mmconfig -i
  f8000000-fbffffff : PCI MMCONFIG 0000 [bus 00-3f]
铁 busybox git:(master) 参[00m sudo ./busbox devmem 0xf8019000
sudo: ./busbox: command not found
铁 busybox git:(master) 参[00m sudo ./busybox devmem 0xf8019000
0x14531022
铁 busybox git:(master) 参[00m lspci -s 0:3.1 -xxx
00:03.1 PCI bridge: Advanced Micro Devices, Inc. [AMD] Family 17h (Models 00h-0fh) PCIe GPP Bridge
00: 22 10 53 14 07 04 10 20 00 00 04 06 10 00 81 00
10: 00 00 00 00 00 00 00 00 00 00 22 22 00 e1 e1 00 20
20: 90 fe 90 fe 01 e0 11 f0 00 00 00 00 00 00 00 00
30: 00 00 00 00 50 00 00 00 00 00 00 00 ff 00 18 00

铁 busybox git:(master) 参[00m sudo ./busybox devmem 0xf8000000
0x14501022
铁 busybox git:(master) 参[00m lspci -s 0:0.0 -xxx
00:00.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 17h (Models 00h-0fh) Root Complex
00: 22 10 50 14 00 00 00 00 00 00 06 00 00 80 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 43 10 47 87
```


BAR(base address registers) 就是为了把设备的内部各种资源映射（芯片内部寄存器或者 DDR）到 IO 空间（IO BAR）或者 memory 空间（memory BAR）。



关于 IO 方式和 memory 方式访问 PCIe 配置空间的具体实现，参考

https://blog.csdn.net/huangkangying/article/details/50570612

https://blog.csdn.net/mao0514/article/details/26072229

https://blog.csdn.net/xingqingly/article/details/45695739

https://zhuanlan.zhihu.com/p/34047690

http://developer.amd.com/wordpress/media/2012/10/pci%20-%20pci%20express%20configuration%20space%20access.pdf

全文完