第Ⅱ篇　PCI Express 体系结构概述

虽然 PCI 总线取得了巨大的成功，但是随着处理器主频的不断提高，PCI 总线提供的带宽愈发显得捉襟见肘。PCI 总线也在不断地进行升级，其位宽和频率从最初的 32 位/33 MHz 扩展到 64 位/66 MHz，而 PCI-X 总线更是将总线频率提高到 533 MHz，能够提供的最大理论带宽为 4263 MB。但是 PCI 总线仍无法解决其体系结构中存在的一些缺陷，它面临着一系列挑战，包括带宽、流量控制和数据传送质量等。

PCI 总线的最高工作频率为 66 MHz，最大位宽为 64 位，从理论上讲，PCI 总线可以提供的最大传输带宽为 532 MB。然而 PCI 总线作为一个共享总线，在其上的所有 PCI 设备必须共享 PCI 总线的带宽。同时由于 PCI 总线的协议开销，导致 PCI 总线可以实际利用的数据带宽远小于其峰值带宽。

PCI 总线采用提高总线位宽和频率的方法增加其传输带宽。但是这种方法从性能价格比的角度来看，并不是最优的。数据总线位宽的提高将直接影响芯片的生产成本，64 位的 PCI 总线接口需要设计者使用更多的芯片引脚，从而导致 64 位的 PCI 总线接口芯片的价格远高于 32 位的 PCI 总线接口芯片。与 32 位 PCI 总线接口相比，设计者还需要使用更多的印制板层数来实现 64 位 PCI 总线接口。

而提高总线频率，除了给硬件工程师带来了一系列信号完整性的问题之外，更直接影响 PCI 总线的负载能力。一条 33 MHz 的 PCI 总线最多可以驱动 10 个负载，而 66 MHz 的 PCI 总线最多只能驱动 4 个负载。因此片面提高 PCI 总线的频率和位宽，并不能有效地提高 PCI 总线的带宽。

除此之外 PCI 总线在设计之初并没有考虑服务质量的问题。有些实时数据采集卡、音频或者视频的多媒体应用需要 PCI 总线提供额定带宽，而 PCI 总线上的设备只能轮流使用 PCI 总线，当一个设备长期占用 PCI 总线时，将阻止其他 PCI 设备使用 PCI 总线，从而影响了 PCI 总线的传送质量。

基于以上几个原因，PCI 总线在某种程度上说并不能完全适应现代处理器系统的需要，而使用 PCIe 总线可以有效解决 PCI 总线存在的一些问题。首先 PCIe 总线可以提供更大的总线带宽，PCIe V3.0 支持的最高总线频率为 4GHz，远高于 PCI-X 总线提供的最高总线频率。其次 PCIe 总线支持虚通路（Virtual Channel，VC）技术，优先级不同的数据报文可以使用不同的虚通路，而每一路虚通路可以独立设置缓冲，从而相对合理地解决了数据传送过程中存在的服务质量问题。

PCIe 总线由若干层次组成，包括事务层、数据链路层和物理层。PCIe 总线使用数据报

文进行数据传递，这些数据报文需要通过 PCIe 总线的这些层次。PCIe 总线的这种数据传递方式与互联网使用 TCP/IP 协议进行数据传递有类似之处。

实际上在互联网中存在的许多概念也存在于 PCIe 总线中，如交换、路由和仲裁机制等，不过两者实现上的最大不同在于前者主要使用软件程序实现其协议栈，而后者使用硬件逻辑实现。

半导体工艺的逐步提高，使得更多的软件算法可以使用硬件逻辑来实现，这给从事集成电路设计的工程师带来了巨大的挑战，因为他们使用 Verilog/VHDL 程序编写的算法，之前是使用 C 或 C ++ 这样的高级语言实现的。

PCIe 总线在系统软件级与 PCI 总线兼容，基于 PCI 总线的系统软件几乎可以不经修改直接移植到 PCIe 总线中。绝大多数 PCI/PCI-X 总线使用的总线事务都被 PCIe 总线保留，而 PCI 设备使用的配置空间也被 PCIe 总线继承。基于 PCI 体系结构的系统编程模型，几乎可以在没有本质变化的前提下，直接在 PCIe 体系结构中使用。

但是从体系系统的角度上看，PCIe 总线还是增加了一些新的特性，其中一些特性不仅仅是称呼上的变化，而且在功能上也得到了增强。如在 PCIe 体系结构中出现的 RC（Root Complex）。RC 的主要功能与 PCI 总线中的 HOST 主桥类似，但是在 HOST 主桥的基础上增加了许多功能。

在不同处理器系统中，RC 的实现方式不同，因此仅仅用 PCIe 总线控制器称呼 RC 是不够的，实际上 PCIe 总线规范对 RC 并没有一个合适的解释。RC 本身也随处理器系统的不同而不同，是一个很模糊的概念。Intel 并没有使用 PCIe 总线控制器，而是使用 RC 管理 PCIe 总线。基于深层次的考虑，在 x86 处理器体系结构中，RC 并不仅仅管理 PCIe 设备的数据访问，而且还包含访问控制、错误处理和虚拟化技术等一系列内容。因此用 PCIe 总线控制器统称 RC，在 x86 处理器体系结构中并不合适。

在 PCIe 总线中，还有一些特性与 PCIe 总线协议的实现相关。与 PCI 总线相比，PCIe 总线使用端到端的连接方式，添加流量控制机制，并对“访问序”做出了进一步优化。虽然从系统软件的角度来看，PCI 总线与 PCIe 总线基本一致。但是从硬件设计的角度来看 PCIe 总线完全不同于 PCI 总线，基于 PCIe 总线各类设备的硬件设计难度远大于基于 PCI 总线的对应设备的设计难度。

目前 PCIe 总线规范依然在迅猛发展，但并不是所有 PCIe 设备都支持这些在 PCIe 总线的最新规范中提及的概念。一般说来，PCIe 总线规范提出的新概念，最先在 x86 处理器系统的 Chipset 和 Intel 设计的 PCIe 设备中出现。

第 4 章　PCIe 总线概述

随着现代处理器技术的发展，在互连领域中，使用高速差分总线替代并行总线是大势所趋。与单端并行信号相比，高速差分信号可以使用更高的时钟频率，使用更少的信号线，完成之前需要许多单端并行数据信号才能达到的总线带宽。

PCI 总线使用并行总线结构，在同一条总线上的所有外部设备共享总线带宽，而 PCIe 总线使用了高速差分总线，并采用端到端的连接方式，因此在每一条 PCIe 链路中只能连接两个设备。这使得 PCIe 与 PCI 总线采用的拓扑结构有所不同。PCIe 总线除了在连接方式上与 PCI 总线不同之外，还使用了一些在网络通信中使用的技术，如支持多种数据路由方式，基于多通路的数据传递方式，和基于报文的数据传送方式，并充分考虑了在数据传送中出现的服务质量 QoS（Quality of Service）问题。

4.1　PCIe 总线的基础知识

与 PCI 总线不同，PCIe 总线使用端到端的连接方式，在一条 PCIe 链路的两端只能各连接一个设备，这两个设备互为数据发送端和数据接收端。PCIe 总线除了总线链路外，还具有多个层次，发送端发送数据时将通过这些层次，而接收端接收数据时也使用这些层次。PCIe 总线使用的层次结构与网络协议栈较为类似。

4.1.1　端到端的数据传递

PCIe 链路使用“端到端的数据传送方式”，发送端和接收端中都含有 TX（发送逻辑）和 RX（接收逻辑），其结构如图 4-1 所示。

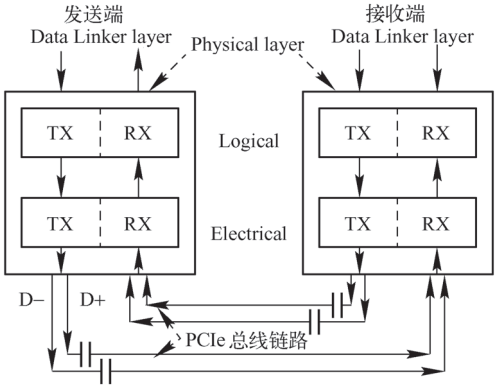

图 4-1　PCIe 总线的物理链路

由上图所示，在 PCIe 总线的物理链路的一个数据通路（Lane）中，有两组差分信号，共 4 根信号线。其中发送端的 TX 部件与接收端的 RX 部件使用一组差分信号连接，该链路也被称为发送端的发送链路，也是接收端的接收链路；而发送端的 RX 部件与接收端的 TX

部件使用另一组差分信号连接，该链路也被称为发送端的接收链路，也是接收端的发送链路。一个 PCIe 链路可以由多个 Lane 组成。

高速差分信号电气规范要求其发送端串接一个电容，以进行 AC 耦合。该电容也被称为 AC 耦合电容。PCIe 链路使用差分信号进行数据传送，一个差分信号由 D + 和 D – 两根信号组成，信号接收端通过比较这两个信号的差值，判断发送端发送的是逻辑“1”还是逻辑“0”。

与单端信号相比，差分信号抗干扰的能力更强，因为差分信号在布线时要求“等长”、“等宽”、“贴近”，而且在同层。因此外部干扰噪声将被“同值”而且“同时”加载到 D + 和 D – 两根信号上，其差值在理想情况下为 0，对信号的逻辑值产生的影响较小。因此差分信号可以使用更高的总线频率。

此外使用差分信号能有效抑制电磁干扰（EMI，Electro Magnetic Interference）。因为差分信号 D + 与 D – 距离很近而且信号幅值相等、极性相反，这两根线与地线间耦合电磁场的幅值相等，将相互抵消，因此差分信号对外界的电磁干扰较小。当然差分信号的缺点也是显而易见的，一是差分信号使用两根信号传送一位数据；二是差分信号的布线相对严格一些。

PCIe 链路可以由多条 Lane 组成，目前 PCIe 链路可以支持 1、2、4、8、12、16 和 32 个 Lane，即 ×1、×2、×4、×8、×12、×16 和 ×32 宽度的 PCIe 链路。每一个 Lane 上使用的总线频率与 PCIe 总线使用的版本相关。

第 1 个 PCIe 总线规范为 V1. 0，之后依次为 V1. 0a，V1. 1，V2. 0 和 V2. 1。目前 PCIe 总线的最新规范为 V2. 1，而 V3. 0 正在开发，预计在 2010 年发布。不同的 PCIe 总线规范所定义的总线频率和链路编码方式并不相同，如表 4–1 所示。

表 4–1　PCIe 总线规范与总线频率和编码的关系

PCIe 总线规范	总线频率①/GHz	单 Lane 的峰值带宽/(GT/s)	编码方式
1. x	1. 25	2. 5	8/10b 编码
2. x	2. 5	5	8/10b 编码
3. 0	4	8	128/130b 编码

① 这里的总线频率指差分信号按照逻辑“0”和“1”连续变化时的频率。

如上表所示，不同的 PCIe 总线规范使用的总线频率并不相同，其使用的数据编码方式也不相同。PCIe 总线 V1. x 和 V2. 0 规范在物理层中使用 8/10b 编码，即在 PCIe 链路上的 10 bit 中含有 8 位的有效数据；而 V3. 0 规范使用 128/130b 编码方式，即在 PCIe 链路上的 130 bit 中含有 128 位的有效数据。

V3. 0 规范使用的总线频率虽然只有 4 GHz，但是其有效带宽是 V2. x 的两倍。有关 8/10 编码的详细描述见第 7. 3. 3 节。下文将以 V2. x 规范为例，说明不同宽度 PCIe 链路所能提供的峰值带宽，如表 4–2 所示。

表 4–2　PCIe 总线的峰值带宽

PCIe 总线的数据位宽	×1	×2	×4	×8	×12	×16	×32
峰值带宽/(GT/s)	5	10	20	40	60	80	160

×32 的 PCIe 链路可以提供 160 GT/s 的链路带宽，远高于 PCI/PCI-X 总线所能提供的峰值带宽。而即将推出的 PCIe V3.0 规范使用 4 GHz 的总线频率，将进一步提高 PCIe 链路的峰值带宽。

在 PCIe 总线中，使用 GT（Gigatransfer）计算 PCIe 链路的峰值带宽。GT 是在 PCIe 链路上传递的峰值带宽，其计算公式为总线频率 × 数据位宽 ×2。

在 PCIe 总线中，影响有效带宽的因素有很多，因而其有效带宽较难计算，这部分内容详见第 12.4.1 节。尽管如此，PCIe 总线提供的有效带宽还是远高于 PCI 总线。PCIe 总线也有其弱点，其中最突出的问题是传送延时。

PCIe 链路使用串行方式进行数据传送，然而在芯片内部，数据总线仍然是并行的，因此 PCIe 链路接口需要进行串并转换，这种串并转换将产生较大的延时。除此之外 PCIe 总线的数据报文需要经过事务层、数据链路层和物理层，这些数据报文在穿越这些层次时，也将带来延时。本书将在第 12.4 节详细讨论 PCIe 总线的延时与带宽之间的联系。

在基于 PCIe 总线的设备中，×1 的 PCIe 链路最为常见，而 ×12 的 PCIe 链路极少出现，×4 和 ×8 的 PCIe 设备也不多见。Intel 通常在 ICH 中集成了多个 ×1 的 PCIe 链路用来连接低速外设，而在 MCH 中集成了一个 ×16 的 PCIe 链路用于连接显卡控制器。而 PowerPC 处理器通常能够支持 ×8、×4、×2 和 ×1 的 PCIe 链路。

PCIe 总线物理链路间的数据传送使用基于时钟的同步传送机制，但是在物理链路上并没有时钟线，PCIe 总线的接收端含有时钟恢复模块 CDR（Clock Data Recovery），CDR 将从接收报文中提取接收时钟，从而进行同步数据传递，PCIe 设备进行链路训练时将完成时钟的提取工作，详见第 8.2 节。

值得注意的是，在一个 PCIe 设备中除了需要从报文中提取时钟外，还使用了REFCLK + 和 REFCLK – 信号对作为本地参考时钟，这个信号对的描述见下文。

4.1.2 PCIe 总线使用的信号

PCIe 设备使用两种电源信号供电，分别是 V_{cc} 与 V_{aux}，其额定电压为 3.3 V。其中 V_{cc} 为主电源，PCIe 设备使用的主要逻辑模块均使用 V_{cc} 供电，而一些与电源管理相关的逻辑使用 V_{aux} 供电。在 PCIe 设备中，一些特殊的寄存器通常使用 V_{aux} 供电，如 Sticky Register，此时即使 PCIe 设备的 V_{cc} 被移除，这些与电源管理相关的逻辑状态和这些特殊寄存器的内容也不会发生改变。

在 PCIe 总线中，使用 V_{aux} 的主要原因是为了降低功耗和缩短系统恢复时间。因为 V_{aux} 在多数情况下并不会被移除，因此当 PCIe 设备的 V_{cc} 恢复后，该设备不必重新恢复使用 V_{aux} 供电的逻辑，从而设备可以很快地恢复到正常工作状状态。

PCIe 链路的最大宽度为 ×32，但是在实际应用中，×32 的链路宽度极少使用。在一个处理器系统中，一般最多提供 ×16 的 PCIe 插槽，并使用 PETp0 ~ 15、PETn0 ~ 15 和 PERp0 ~ 15、PERn0 ~ 15 共 64 根信号线组成 32 对差分信号，其中 16 对 PETxx 信号用于发送链路，另外 16 对 PERxx 信号用于接收链路。这些差分信号的详细说明见第 7.3.1 节。除此之外 PCIe 总线还使用了下列辅助信号。

1. PERST#信号

该信号为全局复位信号，由处理器系统提供，处理器系统需要为 PCIe 插槽和 PCIe 设备

提供该复位信号。PCIe 设备使用该信号复位内部逻辑。当该信号有效时，PCIe 设备将进行复位操作。PCIe 总线定义了多种复位方式，其中 Cold Reset 和 Warm Reset 这两种复位方式的实现与该信号有关，详见第 4.1.5 节。

2. REFCLK + 和 REFCLK – 信号

在一个处理器系统中，可能含有许多 PCIe 设备，这些设备可以作为 Add-In 卡与 PCIe 插槽连接，也可以作为内置模块，与处理器系统提供的 PCIe 链路直接相连，而不需要经过 PCIe 插槽。PCIe 设备与 PCIe 插槽都具有 REFCLK + 和 REFCLK – 信号，其中 PCIe 插槽使用这组信号与处理器系统同步。

在一个处理器系统中，通常采用专用逻辑向 PCIe 插槽提供 REFCLK + 和 REFCLK – 信号，如图 4–2 所示。其中 100 MHz 的时钟源由晶振提供，并经过一个“一推多”的差分时钟驱动器生成多个同相位的时钟源，与 PCIe 插槽一一对应连接。

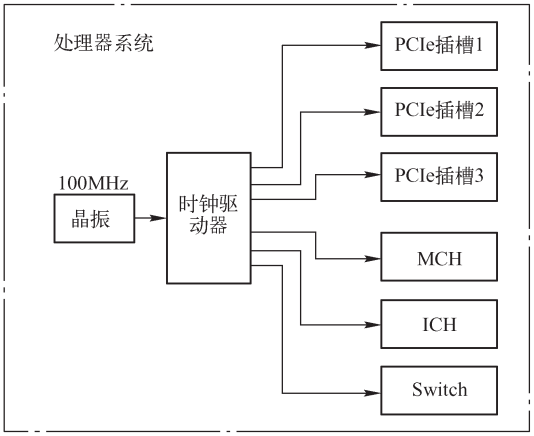

图 4–2　参考时钟与 PCIe 插槽的连接

PCIe 插槽需要使用参考时钟，其频率为 100 MHz。处理器系统需要为每一个 PCIe 插槽、MCH、ICH 和 Switch 提供参考时钟。而且要求在一个处理器系统中，时钟驱动器产生的参考时钟信号到每一个 PCIe 插槽（MCH、ICH 和 Swith）的距离差在 15in 之内。通常信号的传播速度接近光速，约为 6in/ns，由此可见，不同 PCIe 插槽间 REFCLK + 和 REFCLK – 信号的传送延时差约为 2.5 ns。

当 PCIe 设备作为 Add-In 卡连接在 PCIe 插槽时，可以直接使用 PCIe 插槽提供的 REFCLK + 和 REFCLK – 信号，也可以使用独立的参考时钟。内置的 PCIe 设备与 Add-In 卡在处理 REFCLK + 和 REFCLK – 信号时使用的方法类似，PCIe 设备可以使用独立的参考时钟，而不使用 REFCLK + 和REFCLK – 信号。

在 PCIe 设备配置空间的 Link Control Register 中，含有一个“Common Clock Configuration”位。当该位为 1 时，表示该设备与 PCIe 链路的对端设备使用“同相位”的参考时钟；如果为 0，表示该设备与 PCIe 链路的对端设备使用的参考时钟是异步的。Link Control Register 的详细描述见第 4.3.2 节。

在 PCIe 设备中，“Common Clock Configuration”位的缺省值为 0，此时 PCIe 设备使用的

参考时钟与对端设备没有任何联系，PCIe 链路两端设备使用的参考时钟可以异步设置。这个异步时钟设置方法对于使用 PCIe 链路进行远程连接时尤为重要。

在一个处理器系统中，如果使用 PCIe 链路进行机箱到机箱间的互连，因为参考时钟可以异步设置，机箱到机箱之间进行数据传送时仅需要差分信号线即可，而不需要参考时钟，从而极大地降低了连接难度。

3. WAKE#信号

当 PCIe 设备进入休眠状态，主电源已经停止供电时，PCIe 设备使用该信号向处理器系统提交唤醒请求，使处理器系统重新为该 PCIe 设备提供主电源 V_{cc}。在 PCIe 总线中，WAKE#信号是可选的，因此使用 WAKE#信号唤醒 PCIe 设备的机制也是可选的。值得注意的是产生该信号的硬件逻辑必须使用辅助电源 V_{aux}供电。

WAKE#是一个 Open Drain 信号，一个处理器的所有 PCIe 设备可以将 WAKE#信号进行线与后，统一发送给处理器系统的电源控制器。当某个 PCIe 设备需要被唤醒时，该设备首先置 WAKE#信号有效，然后在经过一段延时之后，处理器系统开始为该设备提供主电源 V_{cc}，并使用 PERST#信号对该设备进行复位操作。此时 WAKE#信号需要始终保持为低，当主电源 V_{cc}上电完成之后，PERST#信号也将置为无效并结束复位，WAKE#信号也将随之置为无效，结束整个唤醒过程。

PCIe 设备除了可以使用 WAKE#信号实现唤醒功能外，还可以使用 Beacon 信号实现唤醒功能。与 WAKE#信号实现唤醒功能不同，Beacon 使用 In-band 信号，即差分信号 D + 和 D – 实现唤醒功能。Beacon 信号 DC 平衡，由一组通过 D + 和 D – 信号生成的脉冲信号组成。这些脉冲信号宽度的最小值为 2 ns，最大值为 16 μs。当 PCIe 设备准备退出 L2 状态（该状态为 PCIe 设备使用的一种低功耗状态）时，可以使用 Beacon 信号，提交唤醒请求。

4. SMCLK 和 SMDAT 信号

SMCLK 和 SMDAT 信号与 x86 处理器的 SMBus（System Mangement Bus）相关。SMBus 于 1995 年由 Intel 提出，SMBus 由 SMCLK 和 SMDAT 信号组成。SMBus 源于 I^2C 总线，但是与 I^2C 总线存在一些差异。

SMBus 的最高总线频率为 100 kHz，而 I^2C 总线可以支持 400 kHz 和 2 MHz 的总线频率。此外 SMBus 上的从设备具有超时功能，当从设备发现主设备发出的时钟信号保持低电平超过 35 ms 时，将引发从设备的超时复位。在正常情况下，SMBus 的主设备使用的总线频率最低为 10 kHz，以避免从设备在正常使用过程中出现超时。

在 SMbus 中，如果主设备需要复位从设备时，可以使用这种超时机制。而 I^2C 总线只能使用硬件信号才能实现这种复位操作，在 I^2C 总线中，如果从设备出现错误时，单纯通过主设备是无法复位从设备的。

SMBus 还支持 Alert Response 机制。当从设备产生一个中断时，并不会立即清除该中断，直到主设备向 0b0001100 地址发出命令。

上文所述的 SMBus 和 I^2C 总线的区别还是局限于物理层和链路层上，实际上 SMBus 还含有网络层。SMBus 还在网络层上定义了 11 种总线协议，用来实现报文传递。

SMBus 在 x86 处理器系统中得到了大规模普及，其主要作用是管理处理器系统的外部设备，并收集外设的运行信息，特别是一些与智能电源管理相关的信息。PCI 和 PCIe 插槽也为 SMBus 预留了接口，以便 PCI/PCIe 设备与处理器系统进行交互。

在 Linux 系统中，SMBus 得到了广泛的应用，ACPI 也为 SMBus 定义了一系列命令，用于智能电池、电池充电器与处理器系统之间的通信。在 Windows 操作系统中，有关外部设备的描述信息，也是通过 SMBus 获得的。

5. JTAG 信号

JTAG（Joint Test Action Group）是一种国际标准测试协议，与 IEEE 1149.1 兼容，主要用于芯片内部测试。目前绝大多数器件都支持 JTAG 测试标准。JTAG 信号由 TRST#、TCK、TDI、TDO 和 TMS 信号组成。其中 TRST#为复位信号；TCK 为时钟信号；TDI 和 TDO 分别与数据输入和数据输出对应；而 TMS 信号为模式选择。

JTAG 允许多个器件通过 JTAG 接口串联在一起，并形成一个 JTAG 链。目前 FPGA 和 EPLD 可以借用 JTAG 接口实现在线编程（In-System Programming，ISP）功能。处理器也可以使用 JTAG 接口进行系统级调试工作，如设置断点、读取内部寄存器和存储器等一系列操作。除此之外 JTAG 接口也可用作“逆向工程”，分析一个产品的实现细节，因此在正式产品中，一般不保留 JTAG 接口。

6. PRSNT1#和 PRSNT2#信号

PRSNT1#和 PRSNT2#信号与 PCIe 设备的热插拔相关。在基于 PCIe 总线的 Add-In 卡中，PRSNT1#和 PRSNT2#信号直接相连，而在处理器主板中，PRSNT1#信号接地，PRSNT2#信号通过上拉电阻接为高。PCIe 设备的热插拔结构如图 4-3 所示。

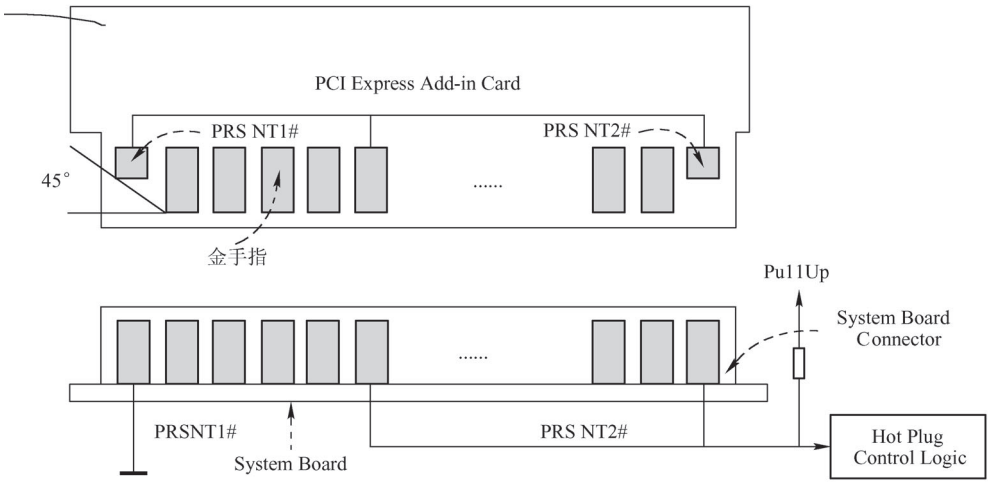

图 4-3　PCIe 设备的热插拔

当 Add-In 卡没有插入时，处理器主板的 PRSNT2#信号由上拉电阻接为高，而当 Add-In 卡插入时主板的 PRSNT2#信号将与 PRSNT1#信号通过 Add-In 卡连通，此时 PRSNT2#信号为低。处理器主板的热插拔控制逻辑将捕获这个“低电平”，得知 Add-In 卡已经插入，从而触发系统软件进行相应处理。

Add-In 卡拔出的工作机制与插入类似。当 Add-In 卡连接在处理器主板时，处理器主板的 PRSNT2#信号为低；当 Add-In 卡拔出后，处理器主板的 PRSNT2#信号为高。处理器主板的热插拔控制逻辑将捕获这个“高电平”，得知 Add-In 卡已经被拔出，从而触发系统软件进行相应处理。

不同的处理器系统处理 PCIe 设备热拔插的过程并不相同，在一个实际的处理器系统中，热拔插设备的实现也远比图 4-3 中的示例复杂得多。值得注意的是，在实现热拔插功能时，Add-In 卡需要使用“长短针”结构。

如图 4-3 所示，PRSNT1#和 PRSNT2#信号使用的金手指长度是其他信号的一半。因此当 PCIe 设备插入插槽时，PRSNT1#和 PRSNT2#信号在其他金手指与 PCIe 插槽完全接触，并经过一段延时后，才能与插槽完全接触；当 PCIe 设备从 PCIe 插槽中拔出时，这两个信号首先与 PCIe 插槽断开，再经过一段延时后，其他信号才能与插槽断开。系统软件可以使用这段延时，进行一些热拔插处理。

4.1.3 PCIe 总线的层次结构

PCIe 总线采用了串行连接方式，并使用数据包（Packet）进行数据传输，采用这种结构有效去除了在 PCI 总线中存在的一些边带信号，如 INTx 和 PME#等信号。在 PCIe 总线中，数据报文在接收和发送过程中，需要通过多个层次，包括事务层、数据链路层和物理层。PCIe 总线的层次结构如图 4-4 所示。

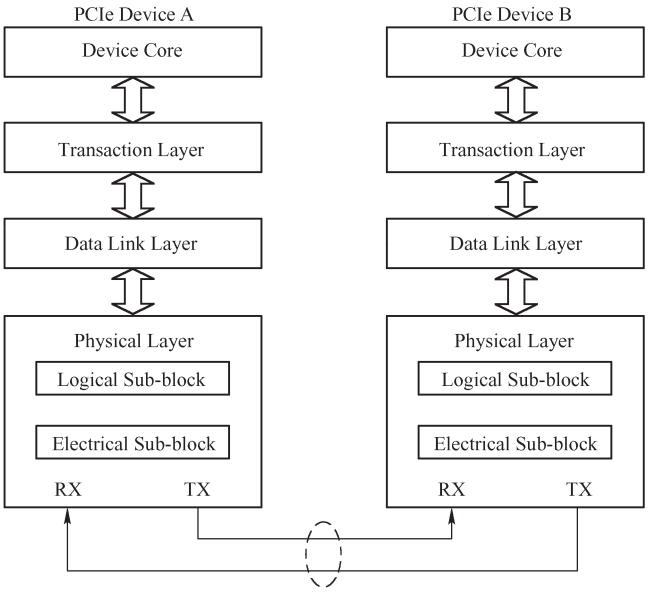

图 4-4　PCI Express 总线的层次组成结构

PCIe 总线的层次组成结构与网络中的层次结构有类似之处，但是 PCIe 总线的各个层次都是使用硬件逻辑实现的。在 PCIe 体系结构中，数据报文首先在设备的核心层（Device Core）中产生，然后再经过该设备的事务层（Transaction Layer）、数据链路层（Data Link Layer）和物理层（Physical Layer），最终发送出去。而接收端的数据也需要通过物理层、数据链路和事务层，并最终到达核心层。

1. 事务层

事务层定义了 PCIe 总线使用总线事务，其中多数总线事务与 PCI 总线兼容。这些总线事务可以通过 Switch 等设备传送到其他 PCIe 设备或者 RC。RC 也可以使用这些总线事务访问 PCIe 设备。

事务层接收来自 PCIe 设备核心层的数据，并将其封装为 TLP（Transaction Layer Packet）后，发向数据链路层。此外事务层还可以从数据链路层中接收数据报文，然后转发至 PCIe 设备的核心层。

事务层的一个重要工作是处理 PCIe 总线的“序”。在 PCIe 总线中，“序”的概念非常重要，也较难理解。在 PCIe 总线中，事务层传递报文时可以乱序，这为 PCIe 设备的设计制造了不小的麻烦。事务层还使用流量控制机制保证 PCIe 链路的使用效率。有关事务层的详细说明见第 6 章。

2. 数据链路层

数据链路层保证来自发送端事务层的报文可以可靠、完整地发送到接收端的数据链路层。来自事务层的报文在通过数据链路层时，将被添加 Sequence Number 前缀和 CRC 后缀。数据链路层使用 ACK/NAK 协议保证报文的可靠传递。

PCIe 总线的数据链路层还定义了多种 DLLP（Data Link Layer Packet），DLLP 产生于数据链路层，终止于数据链路层。值得注意的是，TLP 与 DLLP 并不相同，DLLP 并不是由 TLP 加上 Sequence Number 前缀和 CRC 后缀组成的。数据链路层的详细描述见第 7 章。

3. 物理层

物理层是 PCIe 总线的最底层，将 PCIe 设备连接在一起。PCIe 总线的物理电气特性决定了 PCIe 链路只能使用端到端的连接方式。PCIe 总线的物理层为 PCIe 设备间的数据通信提供传送介质，为数据传送提供可靠的物理环境。

物理层是 PCIe 体系结构最重要，也是最难以实现的组成部分。PCIe 总线的物理层定义了 LTSSM（Link Training and Status State Machine）状态机，PCIe 链路使用该状态机管理链路状态，并进行链路训练、链路恢复和电源管理。

PCIe 总线的物理层还定义了一些专门的“序列”，有的书籍将物理层这些“序列”称为 PLP（Physical Layer Packet），这些序列用于同步 PCIe 链路，并进行链路管理。值得注意的是 PCIe 设备发送 PLP 与发送 TLP 的过程有所不同。

对于系统软件而言，物理层几乎不可见，但是系统程序员仍有必要较为深入地理解物理层的工作原理。本书将在第 7.3 节详细描述 PCIe 总线物理层的实现机制，并在第 8 章详细介绍 LTSSM 状态机。

4.1.4 PCIe 链路的扩展

PCIe 链路使用端到端的数据传送方式。在一条 PCIe 链路中，这两个端口是完全对等的，分别连接发送与接收设备，而且一个 PCIe 链路的一端只能连接一个发送设备或者接收设备。因此 PCIe 链路必须使用 Switch 扩展 PCIe 链路后，才能连接多个设备。使用 Switch 进行链路扩展的实例如图 4-5 所示。

在 PCIe 总线中，Switch[㊀]是一个特殊的设备，该设备由 1 个上游端口和 2 ~ n 个下游端口组成。PCIe 总线规定，在一个 Switch 中可以与 RC 直接或者间接相连[㊁]的端口为上游端

㊀ 本节出现的 Switch 指传统的 Switch，在 MR-IOV 规范定义的 Switch 与此并不相同，详见第 13.3.2 节。

㊁ 所谓间接相连是指通过其他 Switch 再与 RC 相连。

口，在 PCIe 总线中，RC 的位置一般在上方，这也是上游端口这个称呼的由来。在 Switch 中除了上游端口外，其他所有端口都被称为下游端口。下游端口一般与 EP 相连，或者连接下一级 Switch 继续扩展 PCIe 链路。其中与上游端口相连的 PCIe 链路被称为上游链路，与下游端口相连的 PCIe 链路被称为下游链路。

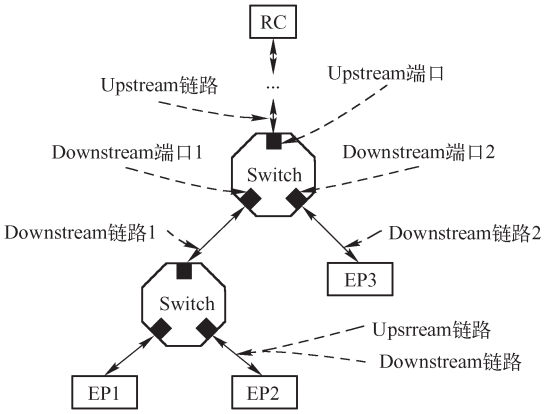

图 4-5　使用 Switch 扩展 PCIe 链路

上游链路和下游链路是相对的概念。如图 4-5 所示，Switch 与 EP2 连接的 PCIe 链路，对于 EP2 而言是上游链路，而对 Switch 而言是下游链路。

图中，Switch 中含有 3 个端口，其中一个是上游端口（Upstream Port），而其他两个为下游端口（Downstream Port）。其中上游端口与 RC 或者其他 Switch 的下游端口相连，而下游端口与 EP 或者其他 Switch 的上游端口相连。

在 Switch 中，还有两个与端口相关的概念，分别是 Egress 端口和 Ingress 端口。这两个端口与通过 Switch 的数据流向有关。其中 Egress 端口指发送端口，即数据离开 Switch 使用的端口；Ingress 端口指接收端口即数据进入 Switch 使用的端口。

Egress 端口和 Ingress 端口与上下游端口没有对应关系。在 Switch 中，上下游端口可以作为 Egress 端口，也可以作为 Ingress 端口。如图 4-5 所示，RC 对 EP3 的内部寄存器进行写操作时，Switch 的上游端口为 Ingress 端口，而下游端口为 Egress 端口；当 EP3 对主存储器进行 DMA 写操作时，该 Switch 的上游端口为 Egress 端口，而下游端口为 Ingress 端口。

PCIe 总线还规定了一种特殊的 Switch 连接方式，即 Crosslink 连接模式。支持这种模式的 Switch，其上游端口可以与其他 Switch 的上游端口连接，其下游端口可以与其他 Switch 的下游端口连接。

PCIe 总线提供 CrossLink 连接模式的主要目的是为了解决不同处理器系统之间的互连，如图 4-6 所示。使用 CrossLink 连接模式时，虽然从物理结构上看，一个 Switch 的上/下游端口与另一个 Switch 的上/下游端口直接相连，但是这个 PCIe 链路经过训练后，仍然是一个端口作为上游端口，而另一个端口作为下游端口。

处理器系统 1 与处理器系统 2 间的数据交换可通过 Crosslink 进行。当处理器系统 1(2) 访问的 PCI 总线域的地址空间或者 Requester ID 不在处理器系统 1(2) 内时，这些数据将被 Crosslink 端口接收，并传递到对端处理器系统中。Crosslink 对端接口的 P2P 桥将接收来自另一个处理器域的数据请求，并将其转换为本处理器域的数据请求。

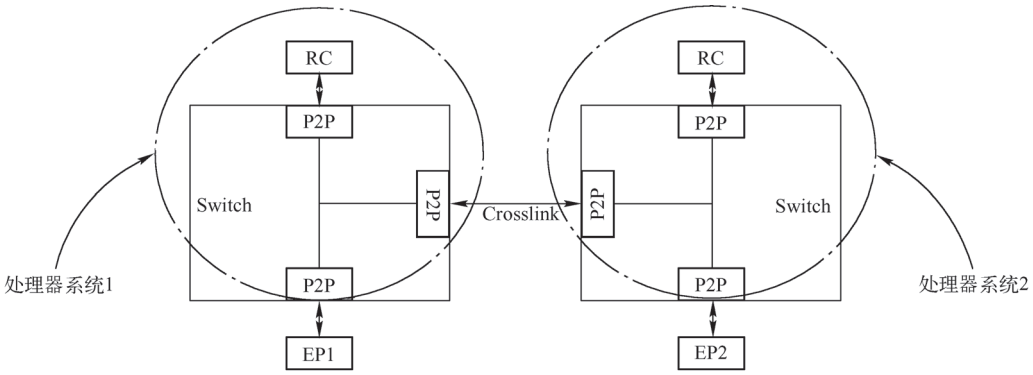

图 4-6　使用 CrossLink 方式连接两个处理器系统

使用 Crosslink 方式连接两个拓扑结构完全相同的处理器系统时，仍然有不足之处。假设图 4-6 中的处理器系统 1 和 2 的 RC 使用的 ID 号都为 0，而主存储器都是从 0x0000 - 0000 开始编址。当处理器 1 读取 EP2 的某段 PCI 总线空间时，EP2 将使用 ID 路由方式，将完成报文传送给 ID 号为 0 的 PCI 设备，此时是处理器 2 的 RC 而不是处理器 1 的 RC 收到 EP2 的数据。因为处理器 1 和 2 的 RC 使用的 ID 号都为 0，EP2 不能区分这两个 RC。

综上所述，使用 Crosslink 方式并不能完全解决两个处理器系统的互连问题，因此在有些 Switch 中支持非透明桥结构。这种结构与 PCI 总线非透明桥的实现机制类似，本章对此不做进一步说明。

使用非透明桥仅解决了两个处理器间数据通路问题，但是不便于 NUMA 结构对外部设备的统一管理。PCIe 总线对此问题的解决方法是使用 MR-IOV 技术，该技术要求 Switch 具有多个上游端口分别与不同的 RC 互连。目前 PLX 公司已经可以提供具有多个上游端口的 Switch，但是尚未实现 MR-IOV 技术涉及的一些与虚拟化相关的技术。有关 MR-IOV 技术的详细说明见第 13. 3. 2 节。

即便 MR-IOV 技术可以合理解决多个处理器间的数据访问和对 PCIe 设备的配置管理，使用 PCIe 总线进行两个或者多个处理器系统间的数据传递仍然是一个不小问题。因为 PCIe 总线的传送延时仍然是制约其在大规模处理器系统互连中应用的重要因素。

4. 1. 5　PCIe 设备的初始化

PCIe 总线规定了两大类复位方式，一种是传统的复位方式（Conventional Reset），另一种是 FLR（Function-Level Reset）方式。

其中 PCIe 总线的传统复位方式由两大类组成，一个是 Fundamental Reset，而另一个是 Non-Fundamental Reset。Fundamental Reset 方式包括 Cold 和 Warm Reset 方式，可以将 PCIe 设备中的绝大多数内部寄存器和内部状态都恢复成初始值；而 Non-Fundamental Reset 方式指 Hot Reset 方式。

1. 传统复位方式

传统复位方式分为 Cold、Warm 和 Hot Reset。PCIe 设备可以根据当前设备的运行状态选择合适的复位方式，PCIe 总线提供多种复位方式的主要原因是减小 PCIe 设备的复位延时。其中传统复位方式的延时大于 FLR 方式。使用传统复位方式时，Cold Reset 使用的时间最

长，而 Hot Reset 使用的时间最短。

当一个 PCIe 设备的 V_{cc} 电源上电后，处理器系统将置该设备的 PERST#信号为有效，此时将引发 PCIe 设备的复位操作，PCIe 总线将这种复位方式称为 Cold Reset 方式。Cold Reset 无疑是一种彻底的复位方式，这种方式属于 Fundamental Reset。PCIe 设备进行 Cold Reset 时，所有使用 V_{cc} 进行供电的寄存器和 PCIe 端口逻辑将无条件进入初始状态。但是使用这种方式依然无法复位使用 V_{aux} 供电的寄存器和逻辑，这些寄存器和逻辑只能在处理器完全下电时才能被彻底复位。

当 PCIe 设备完成上电过程后，也可能重新进行 Fundamental Reset，这种复位方式也被称为 Warm Reset。PCIe 总线并没有规定 Warm Reset 的具体实现方式。一个 PCIe 设备可以使用 Watchdog 逻辑，对该 PCIe 设备进行复位，这种方式就是 Warm Reset 的一种。Warm Reset 也是一种 Fundamental Reset。

除了 Cold 和 Warm Reset 方式外，PCIe 总线还规定了另一种复位方式，即 Hot Reset 方式。当 PCIe 设备出现某种异常时，可以使用软件手段对该设备进行复位。如系统软件将 Bridge Control Register 的 Secondary Bus Reset 位置 1 时，该桥片将 Secondary 总线上的 PCI/PCIe 设备进行 Hot Reset。对于 PCI 总线，当 Secondary Bus Reset 位置 1 时，Secondary 总线将使用 RST#信号对其下游的 PCI 设备进行复位。而 PCIe 总线将通过 TS1 和 TS2 序列对下游设备进行 Hot Reset。

在 TS1 和 TS2 序列中包含一个 Hot Reset 位。当下游设备收到一个 TS1 和 TS2 序列，而且其 Hot Reset 位为 1 时，下游设备将使用 Hot Reset 方式进行复位操作。有关 TS1 和 TS2 序列的详细描述见第 8.1.1 节。除此之外当数据链路层向事务层报告 DL_Down 时，该设备也将进行 Hot Reset。DL_Down 状态的详细说明见第 7.1.1 节。

Hot Reset 方式并不属于 Fundamental Reset。PCIe 设备进行 Hot Reset 方式时，也可以将 PCIe 设备的多数寄存器和状态恢复为初始值。

当 PCIe 设备完成传统复位方式后，经过一段延时后，将开始进行 PCIe 总线的链路训练，有关链路训练的详细说明见第 8 章。

2. FLR 方式

除了传统复位方式之外，PCIe 总线还提供了 FLR 方式。系统软件填写某些寄存器时，PCIe 设备将使用 FLR 方式进行复位。支持 FLR 方式的 PCIe 设备需要在其 BAR 空间中提供一个寄存器，当系统软件对该寄存器的 Function Level Reset 位写 1 时，PCIe 设备将使用 FLR 方式复位 PCIe 设备的内部逻辑。FLR 方式对于 PCIe 设备是可选的，PCIe 总线规范并没有定义 FLR 方式的具体实现过程，但是定义了 FLR 方式的适用范围。

在一个处理器系统中，如果某个 PCIe 设备出现故障时，系统软件需要停止这个 PCIe 设备的所有 I/O 操作。如一个 PCIe 网卡出现故障时，系统软件需要对这个 PCIe 网卡的“与 PCIe 相关”和“与网络部分相关”的逻辑复位，而 Cold、Warm 和 Hot Reset 无法复位“与网络部分相关”的逻辑。此时需要使用 FLR 方式合理复位这个网卡。有些不支持 FLR 方式的网卡，也可以使用传统方式复位“与网络部分相关”的逻辑。

在一个大规模并行处理器系统中，系统软件使用分区的概念管理所有硬件资源，包括处理器资源，和所有 I/O 资源，在这些 I/O 资源中通常会包含 PCIe 设备。在这种处理器系统中，任务在指定的分区中运行，当这个任务执行完毕后，系统软件需要调整硬件资源分区。

此时受到影响的 PCIe 设备需要使用 FLR 方式复位内部逻辑，以免造成对新分区的资源污染，并保护之前任务的运行结果。

此外当系统软件初始化与某个 PCIe 设备相关的软件协议栈时，也可能需要使用 FLR 方式复位 PCIe 设备内部的逻辑。

由上文所示，PCIe 设备使用的 FLR 方式与传统复位方式有所不同。但是对于一些不支持 FLR 方式的 PCIe 设备，也可以使用传统复位方式实现 FLR 方式。但是采用这种方式，与 FLR 方式相比，PCIe 设备的初始化恢复时间较长。因为传统复位方式几乎复位了所有“与 PCIe 链路相关”的逻辑，而 FLR 方式仅复位部分“与 PCIe 链路相关”的逻辑。在 PCIe 总线中，链路训练与重训练的过程较长。

当 PCIe 设备使用 FLR 方式进行复位时，有些与 PCIe 链路相关的状态和寄存器并不会被复位，如下所示。

- Sticky Registers。与传统复位方式相同，FLR 方式也不能复位这些寄存器，但是系统软件可以对部分 Sticky Registers 进行修改。当 V_{aux} 被移除后，在这些寄存器中保存的数据才会丢失。
- HwInit 类型的寄存器。在 PCIe 设备中，有些配置寄存器的属性为 HwInit，这些寄存器的值由芯片的配置引脚决定，或者在上电复位后从 E^2PROM 中获取。Cold 和 Warm Reset 可以复位这些寄存器，然后从 E^2PROM 中重新获取数据；但是使用 FLR 方式，不能复位这些寄存器。
- 此外还有一些特殊的配置寄存器不能被 FLR 方式复位，如 Max_Payload_Size、RCB 和一些与电源管理、流量控制和链路控制直接相关的寄存器。
- FLR 方式不会影响 LTSSM 状态机。

4.2 PCIe 体系结构的组成部件

PCIe 总线作为处理器系统的局部总线，其作用与 PCI 总线类似，主要目的是为了连接处理器系统中的外部设备，当然 PCIe 总线也可以连接其他处理器系统。在不同的处理器系统中，PCIe 体系结构的实现方法略有不同。但是在大多数处理器系统中，都使用了 RC、Switch 和 PCIe-to-PCI 桥这些基本模块连接 PCIe 和 PCI 设备。在 PCIe 总线中，基于 PCIe 总线的设备，也称为 EP（Endpoint）。

4.2.1 基于 PCIe 架构的处理器系统

在不同的处理器系统中，PCIe 体系结构的实现方式不尽相同。PCIe 体系结构以 Intel 的 x86 处理器为蓝本实现，已深深地烙下 x86 处理器的印记。在 PCIe 总线规范中，有许多内容是 x86 处理器独有的，也仅在 x86 处理器的 Chipset 中存在。在 PCIe 总线规范中，一些最新的功能也在 Intel 的 Chipset 中率先实现。

本节将以一个虚拟的处理器系统 A 和 PowerPC 处理器为例简要介绍 RC 的实现，并简单归纳 RC 的通用实现机制。

1. 处理器系统 A

在有些处理器系统中，没有直接提供 PCI 总线，此时需要使用 PCIe 桥，将 PCIe 链路转

换为 PCI 总线之后，才能连接 PCI 设备。在 PCIe 体系结构中，也存在 PCI 总线号的概念，其编号方式与 PCI 总线兼容。一个基于 PCIe 架构的处理器系统 A 如图 4-7 所示。

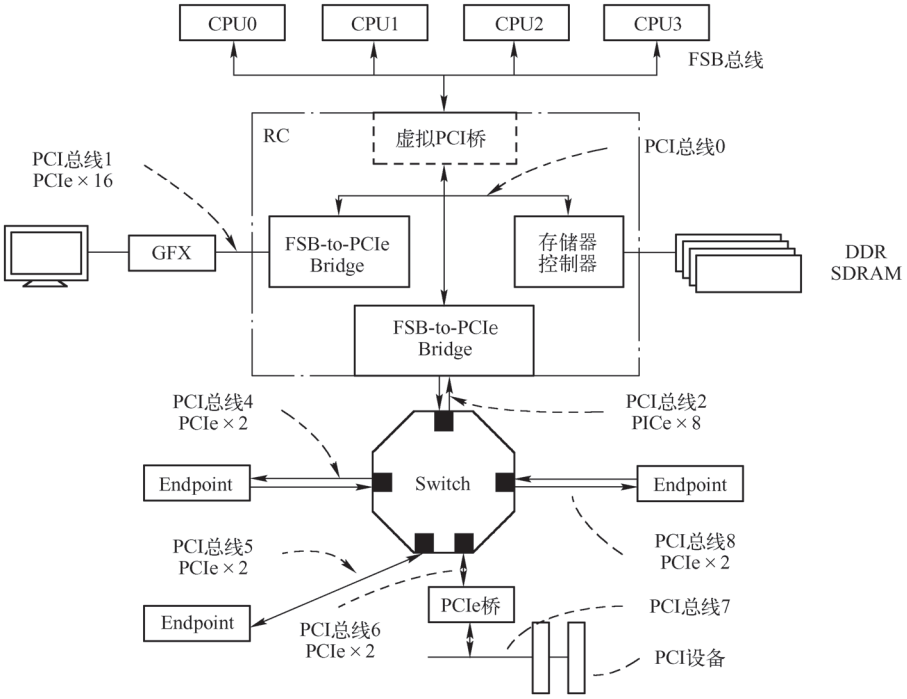

图 4-7　基于 PCIe 总线的处理器系统 A

在图 4-7 的结构中，处理器系统首先使用一个虚拟的 PCI 桥分离处理器系统的存储器域与 PCI 总线域。FSB 总线下的所有外部设备都属于 PCI 总线域。与这个虚拟 PCI 桥直接相连的总线为 PCI 总线 0。这种架构与 Intel 的 x86 处理器系统较为类似。

在这种结构中，RC 由两个 FSB-to-PCIe 桥和存储器控制器组成。值得注意的是在图 4-7 中，虚拟 PCI 桥的作用只是分离存储器域与 PCI 总线域，但是并不会改变信号的电气特性。RC 与处理器通过 FSB 连接，而从电气特性上看，PCI 总线 0 与 FSB 兼容，因此在 PCI 总线 0 上挂接的是 FSB-to-PCIe 桥，而不是 PCI-to-PCIe 桥。

在 PCI 总线 0 上有一个存储器控制器和两个 FSB-to-PCIe 桥。这两个 FSB-to-PCIe 桥分别推出一个 ×16 和 ×8 的 PCIe 链路，其中 ×16 的 PCIe 链路连接显卡控制器（GFX），其编号为 PCI 总线 1； ×8 的 PCIe 链路连接一个 Switch 进行 PCIe 链路扩展。而存储器控制器作为 PCI 总线 0 的一个 Agent 设备，连接 DDR 插槽或者颗粒。

此外在这个 PCI 总线上还可能连接了一些使用“PCI 配置空间”管理的设备，这些设备的访问方法与 PCI 总线兼容，在 x86 处理器的 Chipset 中集成了一些内嵌的设备。这些内嵌的设备均使用“PCI 配置空间”进行管理，包括存储器控制器。

PCIe 总线使用端到端的连接方式，因此只有使用 Switch 才能对 PCIe 链路进行扩展，而每扩展一条 PCIe 链路将产生一个新的 PCI 总线号。如图 4-7 所示，Switch 可以将 1 个 ×8 的 PCIe 端口扩展为 4 个 ×2 的 PCIe 端口，其中每一个 PCIe 端口都可以挂接 EP。除此之外 PCIe 总线还可以使用 PCIe 桥，将 PCIe 总线转换为 PCI 总线或者 PCI-X 总线，之后挂接 PCI/

PCI-X 设备。多数 x86 处理器系统使用这种结构连接 PCIe 或者 PCI 设备。

采用这种结构，有利于处理器系统对外部设备进行统一管理，因为所有外部设备都属于同一个 PCI 总线域，系统软件可以使用 PCI 总线协议统一管理所有外部设备。然而这种外部设备管理方法并非尽善尽美，使用这种结构时，需要注意存储器控制器使用的寄存器也被映射为 PCI 总线空间，从而属于 PCI 总线域，而主存储器（如 DDR 内存空间）仍然属于存储器域。因此在这种结构中，存储器域与 PCI 总线域的划分并不明晰。

在 PCIe 总线规范中并没有明确提及 PCIe 主桥，而使用 RC 概括除了处理器之外的所有与 PCIe 总线相关的内容。在 PCIe 体系结构中，RC 是一个很模糊也很混乱的概念。Intel 使用 PCI 总线的概念管理所有外部设备，包括与这些外部设备相关的寄存器，因此在 RC 中包含一些实际上与 PCIe 总线无关的寄存器。使用这种方式有利于系统软件使用相同的平台管理所有外部设备，也利于平台软件的一致性，但是仍有其不足之处。

PCIe 总线在 x86 处理器中始终处于核心地位。Intel 也借 PCIe 总线统一管理所有外部设备，并以此构建基于 PCIe 总线的 PC 生态系统（Ecosystem）。PCI/PCIe 总线在 x86 处理器系统中的地位超乎想象，而且并不仅局限于硬件层面。

2. PowerPC 处理器

PowerPC 处理器挂接外部设备使用的拓扑结构与 x86 处理器不同。在 PowerPC 处理器中，虽然也含有 PCI/PCIe 总线，但是仍然有许多外部设备并不是连接在 PCI 总线上的。在 PowerPC 处理器中，PCI/PCIe 总线并没有在 x86 处理器中的地位。在 PowerPC 处理器中，还含有许多内部设备，如 TSEC（Three Speed Ethenet Controller）和一些内部集成的快速设备，与 SoC 平台总线直接相连，而不与 PCI/PCIe 总线相连。在 PowerPC 处理器中，PCI/PCIe 总线控制器连接在 SoC 平台总线的下方。

Freescale 即将发布的 P4080 处理器，采用的互连结构与之前的 PowerPC 处理器有较大的不同。P4080 处理器是 Freescale 第一颗基于 E500mc 内核的处理器。E500mc 内核与之前的 E500 V2 和 V1 相比，从指令流水线结构、内存管理和中断处理上说并没有本质的不同。E500mc 内核内置了一个 128 KB 大小的 L2 Cache，该 Cache 连接在 BSB 总线上；而 E500 V1/V2 内核中并不含有 L2 Cache，而仅含有 L1 Cache，而且与 FSB 直接相连。在 E500mc 内核中，还引入了虚拟化的概念。

P4080 处理器共集成了 8 个 E500mc 内核，并使用 CoreNet 连接这 8 个 E500mc 内核，由 CoreNet 互连的处理器使用交换结构进行数据交换，而不是基于共享总线结构。在 P4080 处理器中，一些快速外部设备，如 DDR 控制器、以太网控制器和 PCI/PCIe 总线接口控制器也是直接或者间接地连接到 CoreNet 中，在 P4080 处理器，L3 Cache 也是连接到 CoreNet 中。P4080 处理器的拓扑结构如图 4-8 所示。

目前 Freescale 并没有公开 P4080 处理器的 L1、L2 和 L3 Cache 如何进行 Cache 共享一致性。多数采用 CoreNet 架构互连的处理器系统使用目录表法进行 Cache 共享一致性。但是 P4080 处理器并不是一个追求峰值运算的 SMP 处理器系统，而针对 Data Plane 的应用，因此 P4080 处理器可能并没有使用基于目录表的 Cache 一致性协议。在基于全互连网络的处理器系统中如果使用“类总线监听法”进行 Cache 共享一致性，将不利于多个 CPU 共享同一个存储器系统，在 Cache 一致性的处理过程中容易形成瓶颈。

如图 4-8 所示，P4080 处理器的设计重点并不是 E500mc 内核，而是 CoreNet。CoreNet 内部由全互连网络组成，其中任意两个端口间的通信并不会影响其他端口间的通信。与 MPC8548 处理器相同，P4080 处理器也使用 OceaN[㊀]结构连接 PCIe 与 RapidIO 接口。

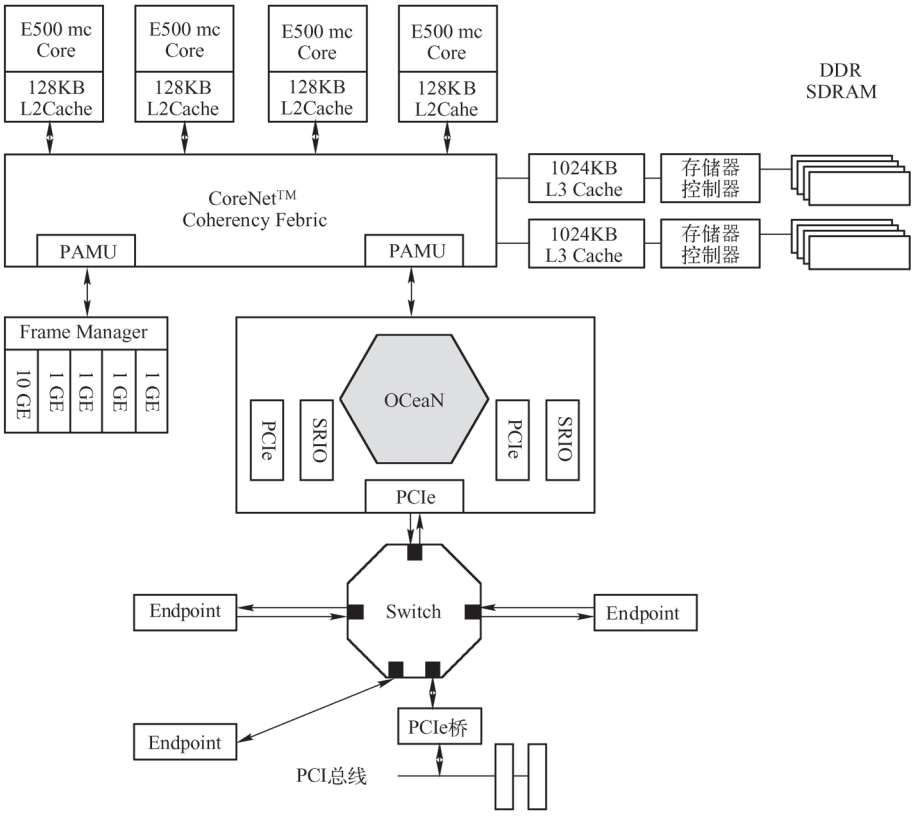

图 4-8　基于 PCIe 总线的 PowerPC 处理器系统

在 P4080 处理器中不存在 RC 的概念，而仅存在 PCIe 总线控制器，当然也可以认为在 P4080 处理器中，PCIe 总线控制器即为 RC。P4080 处理器内部含有 3 个 PCIe 总线控制器，如果该处理器需要连接更多的 PCIe 设备时，需要使用 Switch 扩展 PCIe 链路。

在 P4080 处理器中，所有外部设备与处理器内核都连接在 CoreNet 中，而不使用传统的 SoC 平台总线[㊁]进行连接，从而在有效提高了处理器与外部设备间通信带宽的同时，极大降低了访问延时。此外 P4080 处理器系统使用 PAMU（Peripheral Access Management Unit）分隔外设地址空间与 CoreNet 地址空间。在这种结构下，10 GE/1 GE 接口使用的地址空间与 PCI 总线空间独立。

P4080 处理器使用的 PAMU 是对 MPC8548 处理器 ATMU 的进一步升级。使用这种结构时，外部设备使用的地址空间、PCI 总线域地址空间和存储器域地址空间的划分更加明晰。

㊀ OCeaN 是一个基于交叉矩阵的总线结构，连接在 OCeaN 中的外部设备可以直接通信，而不相互干扰。

㊁ 这种方式也可以被认为是 SoC 平台总线从共享总线结构升级到 Switch 结构。

在 P4080 处理器中，存储器控制器和存储器都属于一个地址空间，即存储器域地址空间。此外这种结构还使用 OCeaN 连接 SRIO[⊖]和 PCIe 总线控制器，使得在 OCeaN 中的 PCIe 端口之间[⊜]可以直接通信，而不需要通过 CoreNet，从而减轻了 CoreNet 的负载。

从内核互连和外部设备互连的结构上看，这种结构具有较大的优势。但是采用这种结构需要使用占用芯片更多的资源，CoreNet 的设计也十分复杂。而最具挑战的问题是，在这种结构之下，Cache 共享一致性模型的设计与实现。

在 Boxboro EX 处理器系统中，可以使用 QPI 将多个处理器系统进行点到点连接，也可以组成一个全互连的处理器系统。这种结构与 P4080 处理器使用的结构类似，但是 Boxboro EX 处理器系统包含的 CPU 更多。

这种全互连的处理器结构也许是未来多核处理器发展的趋势，但是在没有合理地解决多核处理器可编程性问题之前，这种结构很可能不会被系统软件高效地利用，这也是这一结构所面临的挑战。

3. 基于 PCIe 总线的通用处理器结构

在不同的处理器系统中，RC 的实现有较大差异。PCIe 总线规范并没有规定 RC 的实现细则。在有些处理器系统中，RC 相当于 PCIe 主桥，也有的处理器系统也将 PCIe 主桥称为 PCIe 总线控制器。而在 x86 处理器系统中，RC 除了包含 PCIe 总线控制器之外，还包含一些其他组成部件，因此 RC 并不等同于 PCIe 总线控制器。

如果一个 RC 可以提供多个 PCIe 端口，这种 RC 也被称为多端口 RC。如 MPC8572 处理器的 RC 可以直接提供 3 条 PCIe 链路，因此可以直接连接 3 个 EP。如果 MPC8572 处理器需要连接更多 EP 时，需要使用 Switch 进行链路扩展。

而在 x86 处理器系统中，RC 并不是存在于一个芯片中，如在 Montevina 平台中，RC 由 MCH 和 ICH 两个芯片组成。有关 Montevina 平台的详细说明见第 5 章。本节并不对 x86 和 PowerPC 处理器使用的 PCIe 总线结构做进一步讨论，而只介绍这两种结构的相同之处。一个通用的、基于 PCIe 总线的处理器系统如图 4-9 所示。

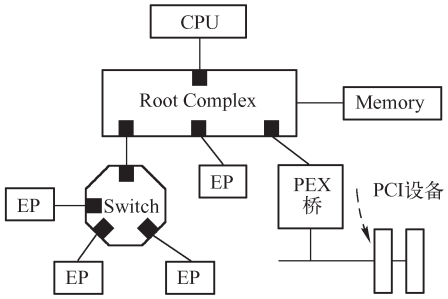

图 4-9　基于 PCIe 总线的通用处理器系统

图中所示的结构将 PCIe 总线端口、存储器控制器等一系列与外部设备有关的接口都集成在一起，并统称为 RC。RC 具有一个或者多个 PCIe 端口，可以连接各类 PCIe 设备。PCIe

⊖ SRIO 为串行 RapidIO。

⊜ PCIe 端口之间的直接通信过程也称为 Peer-to-Peer 传送方式。

设备包括 EP（如网卡、显卡等设备）、Switch 和 PCIe 桥。

PCIe 总线采用端到端的连接方式，每一个 PCIe 端口只能连接一个 EP，当然 PCIe 端口也可以连接 Switch 进行链路扩展。通过 Switch 扩展出的 PCIe 链路可以继续挂接 EP 或者其他 Switch。

4.2.2 RC 的组成结构

RC 是 PCIe 体系结构的一个重要组成部件，也是一个较为混乱的概念。RC 的提出与 x86 处理器系统密切相关。事实上，只有 x86 处理器才存在 PCIe 总线规范定义的“标准 RC”，而在多数处理器系统中，并不含有在 PCIe 总线规范中涉及的与 RC 相关的全部概念。

不同处理器系统的 RC 设计并不相同，在图 4-7 中的处理器系统中，RC 包括存储器控制器、两个 FSB-to-PCIe 桥。而在图 4-8 中的 PowerPC 处理器系统中，RC 的概念并不明晰。在 PowerPC 处理器中并不存在真正意义上的 RC，而仅包含 PCIe 总线控制器。

在 x86 处理器系统中，RC 内部集成了一些 PCI 设备、RCRB（RC Register Block）和 Event Collector 等组成部件。其中 RCRB 由一系列“管理存储器系统”的寄存器组成，而仅存在于 x86 处理器中；而 Event Collector 用来处理来自 PCIe 设备的错误消息报文和 PME 消息报文。RCRB 寄存器组属于 PCI 总线域地址空间，x86 处理器访问 RCRB 的方法与访问 PCI 设备的配置寄存器相同。在有些 x86 处理器系统中，RCRB 在 PCI 总线 0 的设备 0 中。

RCRB 是 x86 处理器所独有的，PowerPC 处理器也含有一组“管理存储器系统”的寄存器，这组寄存器与 RCRB 所实现的功能类似。但是在 PowerPC 处理器中，该组寄存器以 CCSRBAR 寄存器为基地址，处理器采用存储器映像方式访问这组寄存器。

如果将 RC 中的 RCRB、内置的 PCI 设备和 Event Collector 去除，该 RC 的主要功能与 PCI 总线中的 HOST 主桥类似，其主要作用是完成存储器域到 PCI 总线域的地址转换。但是随着虚拟化技术的引入，尤其是引入 MR-IOV 技术之后，RC 的实现变得异常复杂。有关 MR-IOV 技术的详细说明见第 13.3.2 节。

但是 RC 与 HOST 主桥并不相同，RC 除了完成地址空间的转换之外，还需要完成物理信号的转换。在 PowerPC 处理器的 RC 中，来自 OCeaN 或者 FSB 的信号协议与 PCIe 总线信号使用的电气特性并不兼容，使用的总线事务也并不相同，因此必须进行信号协议和总线事务的转换。

在 P4080 处理器中，RC 的下游端口可以挂接 Switch 扩展更多的 PCIe 端口，也可以只挂接一个 EP。在 P4080 处理器的 RC 中，设置了一组 Inbound 和 Outbound 寄存器组，用于存储器域与 PCI 总线域之间地址空间的转换；而 P4080 处理器的 RC 还可以使用 Outbound 寄存器组将 PCI 设备的配置空间直接映射到存储器域中。PowerPC 处理器在处理 PCI/PCIe 接口时，都使用这组 Inbound 和 Outbound 寄存器组。

在 P4080 处理器中，RC 可以使用 PEX_CONFIG_ADDR 与 PEX_CONFIG_DATA 寄存器对 EP 进行配置读写，这两个寄存器与 MPC8548 处理器 HOST 主桥的 PCI_CONFIG_ADDR 和 PCI_CONFIG_DATA 寄存器类似，本章不再详细介绍这组寄存器。

而 x86 处理器的 RC 设计与 PowerPC 处理器有较大的不同，实际上和大多数处理器系统都不相同。x86 处理器赋予了 RC 新的含义，PCIe 总线规范中涉及的 RC 也以 x86 处理器为例进行说明，而且一些在 PCIe 总线规范中出现的最新功能也在 Intel 的 x86 处理器系统中率

先实现。在 x86 处理器系统中的 RC 实现也比其他处理器系统复杂得多。深入理解 x86 处理器系统的 RC 对于理解 PCIe 体系结构非常重要，因此本书将以 Montivina 平台为例详细介绍 x86 处理器中的 RC，其详细描述见第 5 章。

4.2.3 Switch

第 4.1.4 节简单介绍了在 PCIe 总线中，如何使用 Switch 进行链路扩展，本节主要介绍 Switch[⊖]的内部结构。从系统软件的角度上看，每一个 PCIe 链路都占用一个 PCI 总线号，但是一条 PCIe 链路只能连接一个 PCI 设备、Switch、EP 或者 PCIe 桥片。PCIe 总线使用端到端的连接方式，一条 PCIe 链路只能连接一个设备。

一个 PCIe 链路需要挂接多个 EP 时，需要使用 Switch 进行链路扩展。一个标准 Switch 具有一个上游端口和多个下游端口。上游端口与 RC 或者其他 Switch 的下游端口相连，而下游端口可以与 EP、PCIe-to-PCI-X/PCI 桥或者下游 Switch 的上游端口相连。

PCIe 总线规范还支持一种特殊的连接方式，即 Crosslink 连接方式。使用这种方式时，Switch 的下游端口可以与其他 Switch 的下游端口直接连接，上游端口也可以其他 Switch 的上游端口直接连接。在 PCIe 总线规范中，Crosslink 连接方式是可选的，并不要求 PCIe 设备一定支持这种连接方式。

在 PCIe 体系结构中，Switch 的设计难度仅次于 RC，Switch 也是 PCIe 体系结构的核心所在。而从系统软件的角度上看，Switch 内部由多个 PCI-to-PCI 桥组成，其中每一个上游和下游端口都对应一个虚拟 PCI 桥。在一个 Switch 中有多少个端口，在其内部就有多少个虚拟 PCI 桥，就有多少个 PCI 桥配置空间。值得注意的是，在 Switch 内部还具有一条虚拟的 PCI 总线，用于连接各个虚拟 PCI 桥，系统软件在初始化 Switch 时，需要为这条虚拟 PCI 总线编号。Switch 的组成结构如图 4-10 所示。

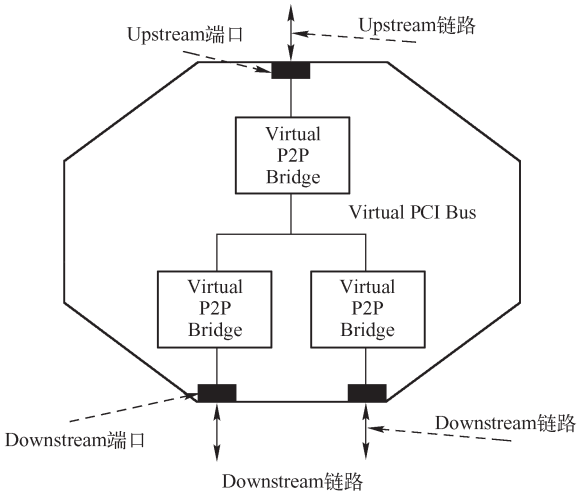

图 4-10　Switch 的等效逻辑图

⊖ PCIe 总线中的 Switch 与网络应用的 Switch 的功能并不相同，而与网络应用中的 Route 功能接近。

Switch 需要处理 PCIe 总线传输过程中的 QoS 问题[㊀]。PCIe 总线的 QoS 要求 PCIe 总线区别对待优先权不同的数据报文，而且无论 PCIe 总线的某一个链路多么拥塞，优先级高的报文，如等时报文（Isochronous Packet）都可以获得额定的数据带宽。而且 PCIe 总线需要保证优先级较高的报文优先到达。PCIe 总线采用虚拟多通路 VC 技术[㊁]，并在这些数据报文中设定一个 TC（Traffic Class）标签，该标签由 3 位组成，将数据报文根据优先权分为 8 类，这 8 类数据报文可以根据需要选择不同的 VC 进行传递。

在 PCIe 总线中，每一条数据链路上最多可以支持 8 个独立的 VC。每个 VC 可以设置独立的缓冲，用来接收和发送数据报文。在 PCIe 体系结构中，TC 和 VC 紧密相连，TC 与 VC 之间的关系是“多对一”。

TC 可以由软件设置，系统软件可以选择某类 TC 由哪个 VC 进行传递。其中一个 VC 可以传递 TC 不相同的数据报文，而 TC 相同的数据报文在指定一个 VC 传递之后，不能再使用其他 VC。在许多处理器系统中，Switch 和 RC 仅支持一个 VC，而 x86 处理器系统和 PLX 的 Switch 中可以支持两个 VC。

下文将以一个简单的例子说明如何使用 TC 标签和多个 VC，以保证数据传送的服务质量。将 PCIe 总线的端到端数据传递过程模拟为使用汽车将一批货物从 A 点运送到 B 点。如果不考虑服务质量，可以使用一辆汽车运送所有这些货物，经过多次往返就可以将所有货物从 A 点运到 B 点。但是这样做会耽误一些需要在指定时间内到达 B 点的货物。有些货物，如一些急救物资、EMS 等其他优先级别较高的货物，必须及时地从 A 点运送到 B 点。这些急救物资的运送应该有别于其他普通物资的运送。

为此首先将不同种类的货物进行分类，将急救物资定义为 TC3 类货物，EMS 定义为 TC2 类货物，平信定义为 TC1 类货物，一般包裹定义为 TC0 类货物，我们最多可以提供 8 种 TC 类标签进行货物分类。

之后我们使用 8 辆汽车，分别是 VC0 ~ VC7 运送这些货物，其中 VC7 的速度最快，而 VC0 的速度最慢。当发生堵车事件时，VC7 优先行驶，VC0 最后行驶。然后我们使用 VC3 运送急救物资，VC2 运送 EMS，VC1 运送平信，VC0 运送包裹，当然使用 VC0 同时运送平信和包裹也是可以的，但是平信或者包裹不能使用一种以上的汽车运送，如平信如果使用了 VC1 运输，就不能使用 VC0。因为 TC 与 VC 的对应关系是“多对一”的关系。

采用这种分类运输的方法，可以做到在 A 点到 B 点带宽有限的情况下，仍然可以保证急救物资和 EMS 可以及时到达 B 点，从而提高了服务质量。

PCIe 总线除了解决数据传送的 QoS 问题之外，还进一步考虑如何在链路传递过程中，使用流量控制机制防止拥塞。PCIe 总线的流量控制机制较为复杂，第 9 章将介绍和流量控制相关的内容。

在 PCIe 体系结构中，Switch 处于核心地位。PCIe 总线使用 Switch 进行链路扩展，在 Switch 中，每一个端口对应一个虚拟 PCI 桥。深入理解 PCI 桥是理解 Switch 软件组成结构的基础。目前 PCIe 总线提出了 MRA-Switch 的概念，这种 Switch 与传统 Switch 有较大的区别，

㊀ 在 PCIe 体系结构中，RC 和 EP 也需要处理 QoS。

㊁ 有关多通路 VC 的详细说明见第 9 章。

有关这部分内容详见第 13.3 节。

4.2.4 VC 和端口仲裁

在 Switch 中存在多个端口，其中来自不同 Ingress 端口的报文可以发向同一个 Egress 端口，因此 Switch 必须要解决端口仲裁和路由选径的问题。所谓端口仲裁指来自不同 Ingress 端口的报文到达同一个 Egress 端口的报文通过顺序，端口仲裁机制如图 4-11 所示。

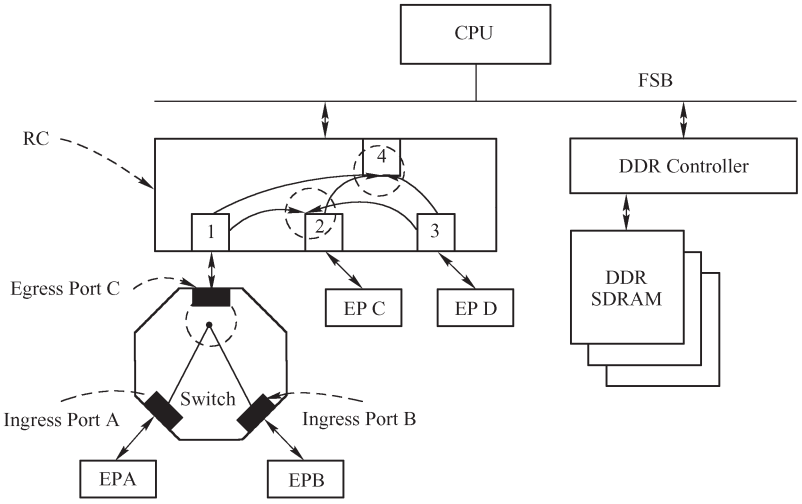

图 4-11　PCIe 总线基于端口的仲裁机制

在一个 Switch 中设有仲裁器，该仲裁器规定了数据报文通过 Switch 的规则。在 PCIe 总线中存在两种仲裁机制，分别是基于 VC 和基于端口的仲裁机制。端口仲裁机制主要针对 RC 和 Switch，当多个 Ingress 端口需要向同一个 Egress 端口发送数据报文时需要进行端口仲裁。具体地讲，在 PCIe 体系结构中有三个端口，需要进行端口仲裁。

- Switch 的 Egress 端口。当 EP A 和 EP B 同时访问 EP C，D 或者 DDR-SDRAM 时，需要通过 Switch 的 Egress 端口 C。此时 Switch 需要进行端口仲裁确定是 EP A 的数据报文还是 EP B 的数据报文优先通过 Egress 端口 C。
- 多端口 RC 的 Egress 端口。当 RC 的端口 1 和端口 3 同时访问 Endpoint C 时，RC 的端口 2 需要进行端口仲裁，决定来自 RC 哪个端口的数据可以率先通过。
- RC 通往主存储器的端口。当 RC 的端口 1、端口 2 和端口 3 同时访问 DDR 控制器时，这些数据报文将通过 RC 的 Egress 端口 4，此时需要进行端口仲裁。

在 PCIe 体系结构中，链路的端口仲裁需要根据每一个 VC 独立设置，而且可以使用不同的算法进行端口仲裁。

下文以图 4-11 中，Switch 的两个 Ingress 端口 A 和 B 向 Egress 端口 C 发送数据报文为例，简要说明端口仲裁和 VC 仲裁的使用方法，其过程如图 4-12 所示。

基于 VC 的仲裁是指发向同一个端口的数据报文，根据使用的 VC 而进行仲裁的方式。当来自端口 B 和端口 A 数据报文（分别使用 VC0 和 VC1 通路）在到达端口 C 之前，需要首先进行端口仲裁后，才能进行 VC 仲裁。PCIe 总线规定了 3 种 VC 仲裁方式，分别为 Strict Priority，RR（Round Robin）和 WRR（Weighted Round Robin）算法。

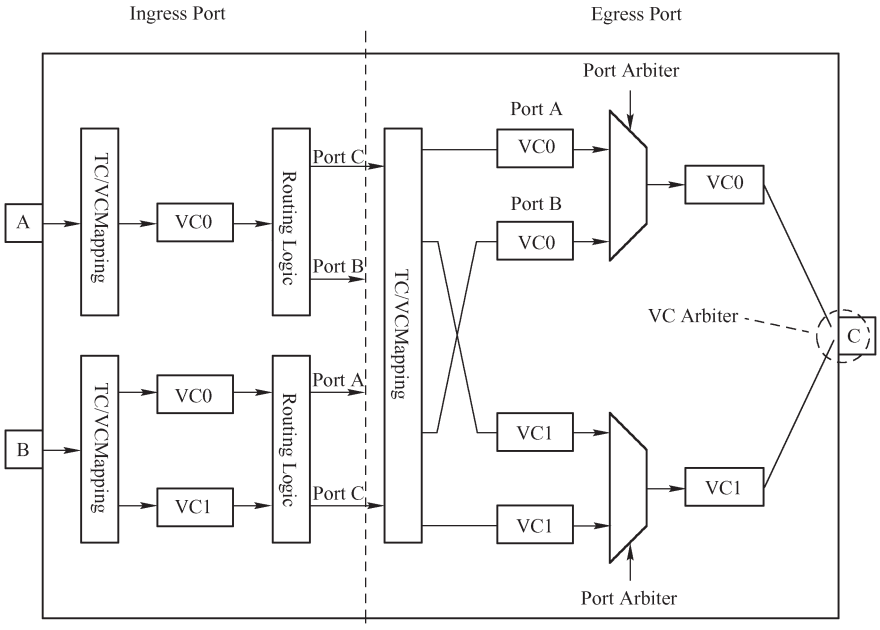

图 4-12　VC 仲裁示意图

当使用 Strict Priority 仲裁方式时，发向 VC7 的数据报文具有最高的优先级，而发向 VC0 的数据报文优先级最低。PCIe 总线允许对 Switch 或者 RC 的部分 VC 采用 Strict Priority 方式进行仲裁，而对其他 VC 采用 RR 和 WRR 算法，如 VC7 ~ VC4 采用 Strict Priority 方式，而采用其他方式处理 VC3 ~ VC0。

使用 RR 方式时，所有 VC 具有相同的优先级，所有 VC 轮流使用 PCIe 链路。WRR 方式与 RR 算法类似，但是可以对每一个 VC 进行加权处理，采用这种方式可以适当提高 VC7 的优先权，而将 VC0 的优先权适当降低。

我们假定 Ingress 端口 A 和 Ingress B 向 Egress 端口 C 进行数据传递时，使用两个 VC 通路，分别是 VC0 和 VC1。其中标签为 TC0 ~ TC3 的数据报文使用 VC0 传送，而标签为 TC4 ~ TC7 数据报文使用 VC1 传送。

而数据报文在离开 Egress 端口 C 时，需要首先进行端口仲裁，之后再通过 VC 仲裁，决定哪个报文优先传送。数据报文从 Ingress A/B 端口发送到 Egress C 端口时，将按照以下步骤进行处理。

（1）首先到达 Ingress A/B 端口的数据报文，将根据该端口的 TC/VC 映射表[㊀]决定使用该端口的哪个 VC 通道。如图 4-12 所示，假设发向端口 A 的数据报文使用 TC0 ~ TC3，而发向端口 B 的数据报文使用 TC0 ~ TC7，这些数据报文在端口 A 中仅使用了 VC0 通道，而在端口 B 中使用了 VC0 和 VC1 两个通道。

（2）数据报文在端口中传递时，将通过路由部件（Routing Logic），将报文发送到合适的端口。如图 4-12 所示，端口 C 可以接收来自端口 A 或端口 B 的数据报文。

㊀ 该表存在于 PCI Express Extended Capabilities 结构中，详见第 4.3.3 节。

（3）当数据报文到达端口 C 时，首先需要经过 TC/VC 映射表，确定在端口 C 中使用哪个 VC 通路接收不同类型的数据报文。

（4）对于端口 C，其 VC0 通道可能会被来自端口 A 的数据报文使用，也可能会被来自端口 B 的数据报文使用。因此在 PCIe 的 Switch 中必须设置一个端口仲裁器，决定来自不同数据端口的数据报文如何使用 VC 通路。

（5）数据报文通过端口仲裁后，获得 VC 通路的使用权之后，还需要经过 Switch 中的 VC 仲裁器，将数据报文发送到实际的物理链路中。

PCIe 总线规定，系统设计者可以使用以下三种方式进行端口仲裁。

（1）Hardware-fixed 仲裁策略。如在系统设计时，采用固化的 RR 仲裁方法。这种方法的硬件实现原理较为简单，此时系统软件不能对端口仲裁器进行配置。

（2）WRR 仲裁策略，即加权的 RR 仲裁策略，该算法和 Time-Based WRR 算法的描述见第 4.3.3 节。

（3）Time-Based WRR 仲裁策略，基于时间片的 WRR 仲裁策略，PCIe 总线可以将一个时间段分为若干个时间片（Phase），每个端口占用其中的一个时间片，并根据端口使用这些时间片的多少对端口进行加权的一种方法。使用 WRR 和 Time-Based WRR 仲裁策略，可以在某种程度上提高 PCIe 总线的 QoS。

PCIe 设备的 Capability 寄存器规定了端口仲裁使用的算法，详见第 4.3.3 节。有些 PCIe 设备并没有提供多种端口仲裁算法，可能也并不含有 Capability 寄存器。此时该 PCIe 设备使用 Hardware-fixed 仲裁策略。

4.2.5 PCIe-to-PCI/PCI-X 桥片

本书将 PCIe-to-PCI/PCI-X 桥片简称为 PCIe 桥片。该桥片有两个作用。

- 将 PCIe 总线转换为 PCI 总线，以连接 PCI 设备。在一个没有提供 PCI 总线接口的处理器中，需要使用这类桥片连接 PCI 总线的外设。许多 PowerPC 处理器在提供 PCIe 总线的同时，也提供了 PCI 总线，因此 PCIe-to-PCI 桥片对基于 PowerPC 处理器系统并不是必须的。
- 将 PCI 总线转换为 PCIe 总线（这也被称为 Reverse Bridge），连接 PCIe 设备。一些低端的处理器并没有提供 PCIe 总线，此时需要使用 PCIe 桥将 PCI 总线转换为 PCIe 总线，才能与其他 PCIe 设备互连。这种用法初看比较奇怪，但是在实际应用中，确实有使用这一功能的可能。本节主要讲解 PCIe 桥的第一个作用。

PCIe 桥的一端与 PCIe 总线相连，而另一端可以与一条或者多条 PCI 总线连接。其中可以连接多个 PCI 总线的 PCIe 桥也被称为多端口 PCIe 桥。

PCIe 总线规范提供了两种多端口 PCIe 桥片的扩展方法。多端口 PCIe 桥片指具有一个上游端口和多个下游端口的桥片。其中上游端口连接 PCIe 链路，而下游端口推出 PCI 总线，连接 PCI 设备。这种桥片的结构如图 4-13 所示。

PCIe 总线规范并没有强制厂商实现多端口 PCIe 桥的办法。但是值得注意的是，使用右图扩展多条 PCI 总线时，在多端口 PCIe 桥中包含一个虚拟的 PCI 总线，即 Bus 2。系统软件对 PCI 总线进行深度优先搜索 DFS（Depth-First Search）时，对左图和右图的处理有些区别。目前多端口 PCIe 桥多使用右图进行端口扩展。

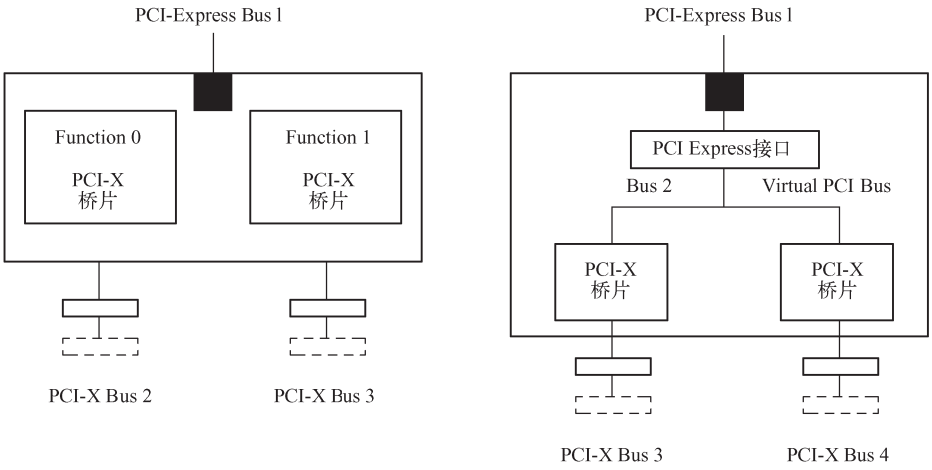

图 4-13　多端口 PCIe 桥的扩展方法

目前虽然 PCIe 总线非常普及，但是仍然有许多基于 PCI 总线的设计，这些基于 PCI 总线的设计可以通过 PCIe 桥，方便地接入到 PCIe 体系结构中。目前有多家半导体厂商可以提供 PCIe 桥片，如 PLX、NXP、Tundra 和 Intel。就功能的完善和性能而言，Intel 的 PCIe 桥无疑是最佳选择，而 PLX 和 Tundra 的 PCIe 桥在嵌入式系统中得到了广泛的应用。

4.3　PCIe 设备的扩展配置空间

本书在第 2.3.2 节讲述了 PCI 设备使用的基本配置空间。这个基本配置空间共由 64 个字节组成，其地址范围为 0x00 ~ 0x3F，这 64 个字节是所有 PCI 设备必须支持的。事实上，许多 PCI 设备也仅支持这 64 个配置寄存器。

此外 PCI/PCI-X 和 PCIe 设备还扩展了 0x40 ~ 0xFF 这段配置空间，在这段空间主要存放一些与 MSI 或者 MSI-X 中断机制和电源管理相关的 Capability 结构。其中所有能够提交中断请求的 PCIe 设备，必须支持 MSI 或者 MSI-X Capability 结构。

PCIe 设备还支持 0x100 ~ 0xFFF 这段扩展配置空间。PCIe 设备使用的扩展配置空间最大为 4KB，在 PCIe 总线的扩展配置空间中，存放 PCIe 设备所独有的一些 Capability 结构，而 PCI 设备不能使用这段空间。

在 x86 处理器中，使用 CONFIG_ADDRESS 寄存器与 CONFIG_DATA 寄存器访问 PCIe 配置空间的 0x00 ~ 0xFF，而使用 ECAM 方式访问 0x000 ~ 0xFFF 这段空间；而在 PowerPC 处理器中，可以使用 CFG_ADDR 和 CFG_DATA 寄存器访问 0x000 ~ 0xFFF，详见第 2.2 节。

PCI-X 和 PCIe 总线规范要求其设备必须支持 Capabilities 结构。在 PCI 总线的基本配置空间中，包含一个 Capabilities Pointer 寄存器，该寄存器存放 Capabilities 结构链表的头指针。在一个 PCIe 设备中，可能含有多个 Capability 结构，这些寄存器组成一个链表，其结构如图 4-14 所示。

其中每一个 Capability 结构都有唯一的 ID 号，每一个 Capability 寄存器都有一个指针，这个指针指向下一个 Capability 结构，从而组成一个单向链表结构，这个链表的最后一个

Capability 结构的指针为 0。

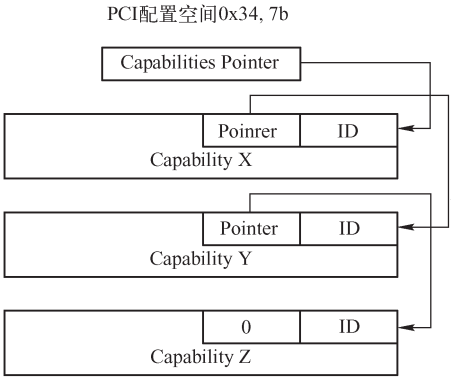

图 4-14　PCIe 总线 Capability 结构的组成

一个 PCIe 设备可以包含多个 Capability 结构，包括与电源管理相关、与 PCIe 总线相关的结构、与中断请求相关的 Capability 结构、PCIe Capability 结构和 PCIe 扩展的 Capability 结构。在本书的其他章节也将讲述这些 Capability 结构，读者在继续其他章节的学习之前，需要简单了解这些 Capability 结构的寄存器组成和使用方法。

其中读者需要重点关注的是 Power Management 和 MSI/MSI-X Capability 结构，Power Management 结构将在本节介绍，而在第 10 章将详细讨论 MSI/MSI-X Capability 结构。在 PCIe 总线规范中，定义了较多的 Capability 结构，这些结构适用于不同的应用场合，在一个指定的 PCIe 设备中，并不一定支持本书涉及的所有 Capability 结构。系统软件程序员也不需要完全掌握 PCIe 总线规范定义的这些 Capability 结构。

4.3.1　Power Management Capability 结构

PCIe 总线使用的软件电源管理机制与 PCI PM（Power Management）兼容。而 PCI 总线的电源管理机制需要使用 Power Management Capability 结构，该结构由一些和 PCI/PCI-X 和 PCIe 总线的电源管理相关的寄存器组成，包括 PMCR（Power Management Capabilities Register）和 PMCSR（Power Management Control and Status Register），其结构如图 4-15 所示。

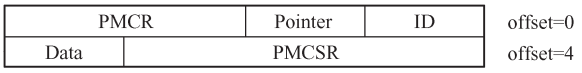

图 4-15　Power Management Capability 结构

Capability ID 字段记载 Power Management Capability 结构的 ID 号，其值为 0x01。在 PCIe 设备中，每一个 Capability 都有唯一的一个 ID 号，而 Next Capability Pointer 字段存放下一个 Capability 结构的地址。

1. PMCR 寄存器

PMCR 寄存器由 16 位组成，其中所有位和字段都是只读的。该寄存器的主要目的是记录当前 PCIe 设备的物理属性，系统软件需要从 PMCR 寄存器中获得当前 PCIe 设备的信息后，才能对 PMCSR 寄存器进行修改。该寄存器的结构如图 4-16 所示，其中 PMCR 寄存器

在 Power Management Capability 结构的第 3 ~ 2 字节中。

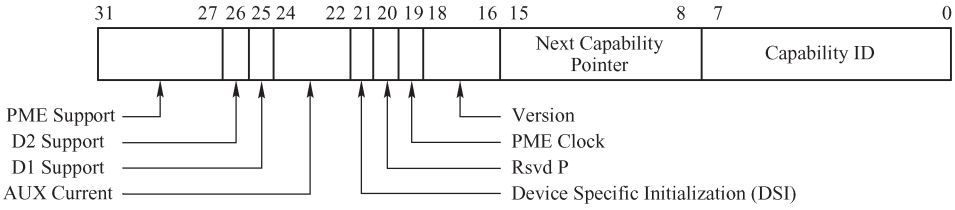

图 4-16　Power Management Capabilities 寄存器

- Version 字段只读，记录 Power Management Capability 结构的版本号。
- PME Clock 位只读，该位没有被 PCIe 总线使用[⊖]，硬件逻辑必须将其接为 0。PCI 设备可以使用 PME#信号通知设备改变电源状态，在 PCI 总线中，如果 PME#信号需要使用时钟（PCI Clock）时，该位为 1；否则该位为 0。PCI 设备改变电源状态时，将 PME#信号置为有效，向处理器系统提交请求。系统软件将这个请求处理完毕后，将通知这个 PCI 设备，之后该 PCI 设备将 PME#信号置为无效。
- RsvdP 字段为系统保留字段。
- DSI（Device Specific Initialization）位只读。某些 PCIe 设备在上电时处于某种工作模式，之后可以通过重新配置运行在其他工作模式中，此时该设备需要使用 DSI 位表示该设备可以使用自定义的电源工作方式。
- AUX（Auxiliary device）Current 字段只读，表示 PCIe 设备需要使用辅助电源的电流强度。PCIe 设备需要使用两种电源，一个是主电源 V_{cc}，另一个是辅助电源 V_{aux}。当 PCIe 设备进入某种节能状态时，主电源将停止供电，而辅助电源需要继续供电。该字段记录 V_{aux} 使用的电流强度，其最大值为 375 mA，最小值为 0，即不使用 V_{aux}。
- D2 和 D1 位只读。D2 位为 1 表示 PCIe 设备支持 D2 状态；D1 位为 1 表示 PCIe 设备支持 D1 状态。PCI PM 机制规定 PCIe 设备可以支持四种状态，分别为 D0 ~ D3 状态。PCIe 设备处于 D0 状态时的功耗最高，处于 D3 状态时最低。多数支持电源管理的 PCIe 设备仅支持 D0 状态和 D3 状态，而 D1 和 D2 状态可选，有关这四种状态的详细说明见第 8.4.1 节。
- PME Support 字段只读，存放 PCIe 设备支持的电源状态。第 27 位为 1 时，表示 PCIe 设备处于 D0 状态时，可以发送 PME 消息；第 28 位为 1 时，表示 PCIe 设备处于 D1 状态时，可以发送 PME 消息；第 29 位为 1 时，表示 PCIe 设备处于 D2 状态时可以发送 PME 消息；第 30 位为 1 时，表示 PCIe 设备处于 $D3_{hot}$ 状态时可以发送 PME 消息；第 31 位为 1 时，表示 PCIe 设备处于 $D3_{cold}$ 状态时可以发送 PME 消息。

2. PMCSR 寄存器

系统软件可以通过操作 PMCSR 寄存器，完成 PCIe 设备电源状态的迁移。该寄存器的结构如图 4-17 所示。

⊖ PCIe 总线使用消息报文（PME Message）进行电源管理，PCIe 设备不支持 PME#信号。

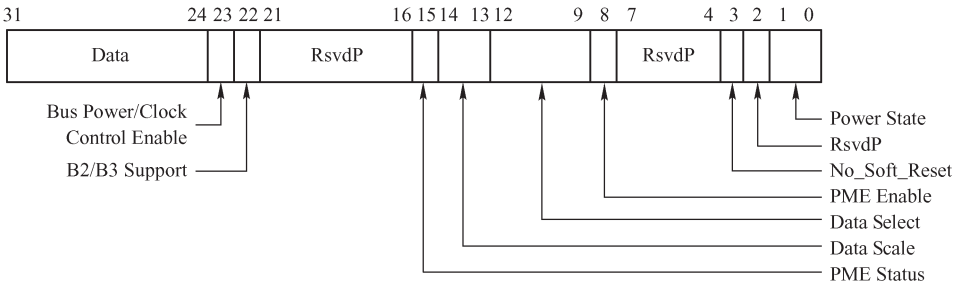

图 4-17　Power Management Status/Control 寄存器

- Power State 字段可读写，该字段记录 PCIe 设备所处的状态。“0b00”与 D0 状态对应；“0b01”与 D1 状态对应；“0b10”与 D2 状态对应；“0b11”与 D3 状态对应。系统软件改变该字段时，PCIe 设备将进行电源状态迁移。
- No_Soft_Reset 位只读。如果该位为 1，PCIe 设备从 $D3_{hot}$ 状态迁移到 D0 状态时，并不需要进行内部复位操作，有关 PCIe 设备配置的现场信息可以由 PCIe 设备的硬件逻辑保存，此时当设备从 D0 状态迁移到 $D3_{hot}$ 状态时，不需要系统软件的干预，其现场由 PCIe 设备主动保存；而该位为 0 时，PCIe 设备从 $D3_{hot}$ 状态迁移到 D0 状态时，需要进行复位操作，因此系统软件在通过改变 Power State 字段使 PCIe 设备从 D0 状态迁移到 $D3_{hot}$ 状态之前，需要保存 PCIe 设备的相关上下文，当 PCIe 设备从 $D3_{hot}$ 状态迁移到 D0 状态时，再进行上下文的恢复操作。
- PME Enable 位，可读写。该位为 1 时，PCIe 设备可以发送 PME 消息；如果为 0，不可以发出 PME 消息。当 PCIe 设备处于 $D3_{cold}$ 状态不能发送 PME 消息时，该位由系统软件设为 0。支持远程唤醒模式的网卡需要将此位使能。
- Data Select 字段可读写，而 Data Scale 字段和 Data 字段只读。系统软件通过这组字段，读取 PCIe 设备处于不同状态时的功耗。首先系统软件置 Data Select 字段为 0 ~ 7 之间的数值，其中 0 和 4 与 D0 状态对应，1 和 5 与 D1 状态对应，2 和 6 与 D2 状态对应，3 和 7 与 D3 状态对应；之后系统软件读取 Data Select 和 Data 字段并以此计算在不同状态下 PCIe 设备的功耗。其中 Data Scale 字段记录精度，为 0 时表示该 PCIe 设备不支持这组字段[⊖]；为 1 时表示 Data 字段的数据需要乘以 0.1 后，才能到得 PCIe 设备的功耗，其单位为 W；为 2 时表示 Data 字段的数据需要乘以 0.01；为 3 时表示 Data 字段的数据需要乘以 0.001。
- PME Status 位，该位只读且写 1 清除，对此位写 0 无意义。该位为 1 时表示 PCIe 设备可以正常发送 PME 消息，系统软件对此位写 1 时，将该位清除。该位由硬件逻辑控制，系统软件仅能清除该位，而不能将该位置 1。
- PCIe 总线没有实现 B2/B3 Support 和 Bus Power/Clock Control Enable 位。在 PCI 总线中，Bus Power/Clock Control Enable 位为 1 时使能 PCI 总线的电源和时钟管理，为 0 时表示关闭；当 Bus Power/Clock Control Enable 位为 1 时，B2/B3 Support 位才有意

⊖ 如果 Data 字段为 0，也表示该 PCIe 设备不支持功耗的测量。

义，B2/B3 Support 位为 1 时表示，当 PCI 桥片处于 $D3_{hot}$ 状态时，这个桥片将停止为 Secondary PCI 总线提供时钟；为 0 时表示，当 PCI 桥片处于 $D3_{hot}$ 状态时，将停止为 Secondary PCI 总线提供电源。

4.3.2 PCI Express Capability 结构

PCI Express Capability 结构存放一些和 PCIe 总线相关的信息，包括 PCIe 链路和插槽的信息。有些 PCIe 设备不一定实现了 PCI Express Capability 结构中的所有寄存器，或者并没有提供这些配置寄存器供系统软件访问。

PCI Express Capability 结构的部分寄存器及其相应字段与硬件的具体实现细节相关，本节仅介绍其中一些系统软件程序员需要了解的字段。在该结构中，Cap ID 字段为 PCI Expresss Capability 结构使用的 ID 号，其值为 0x10。而 Next Capability 字段存放下一个 Capability 寄存器的地址。PCI Express Capability 结构由 PCI Express Capability、Device Capability、Device Control、Device Status、Link Capabilities、Link Status、Link Control、Slot Capabilities 和 Slot Status 等一系列寄存器组成。本节仅介绍该结构中常用的寄存器。PCI Express Capability 的组成结构如图 4-18 所示。

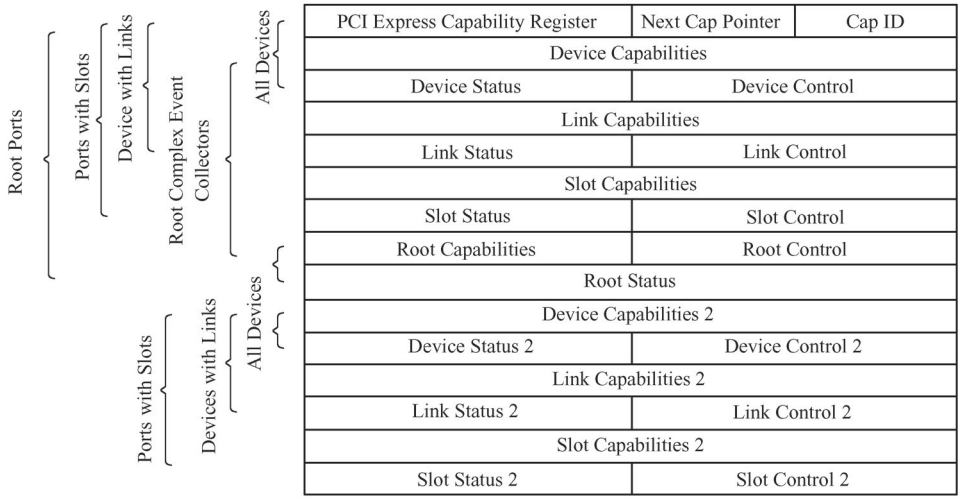

图 4-18　PCI Express Capability 结构的组成结构

1. PCI Express Capability 寄存器

PCI Express Capability 寄存器存放与 PCIe 设备相关的一些参数，包括版本号信息、端口描述，当前 PCIe 链路是与 PCIe 插槽直接连接还是作为内置的 PCIe 设备等一系列信息。这些参数的详细定义如表 4-3 所示。

表 4-3　PCI Express Capability 寄存器

Bits	定　义	描　述
3：0	Capability Version	存放 PCIe 设备的版本号，如果该设备基于 PCIe 总线规范 2. x，该字段的值为 0x2；如果该设备基于 PCIe 总线规范 1. x，该字段的值为 0x1。该字段只读

（续）

Bits	定　义	描　述
7：4	Device/Port Type	存放 PCIe 设备的属性。0b0000 对应 PCIe 总线的 EP；0b0001 对应 Legacy PCIe 总线的 EP；0b0100 对应 RC 的 Root port；0b0101 对应 Switch 的上游端口；0b0110 对应 Switch 的下游端口；0b0111 对应 PCIe 桥片；0b1000 对应 PCI/PCI-X-to-PCIe 桥片；0b1001 对应 RC 中集成的 EP；0b1010 对应 RC 中的 Event Collector①。该字段只读
8	Slot Implemented	当该位为 1 时，表示和当前端口相连的是一个 PCIe 插槽，而不是 PCIe 设备
13：9	Interrupt Message Number	当 PCI Express Capability 结构的 Slot Status 寄存器或者 Root Status 寄存器的状态发生变化时，该 PCIe 设备可以通过 MSI/MSI-X 中断机制向处理器提交中断请求。该字段存放 MSI/MSI-X 中断机制需要的 Message Data 字段。有关 MSI 中断机制的详细描述见第 10 章

① Event Collector 是 RC 集成的一个功能部件，进行错误检查和处理 PME 消息，该部件可选。

2. Device Capability 寄存器

该寄存器的第 2：0 字段为“Max_Payload_Size Supported”字段，该字段存放该设备支持的 Max_Payload_Size 参数的大小，该字段只读，如表 4-4 所示。

表 4-4　PCIe 设备支持的 Max_Payload_Size

Bit［2：0］	支持的 Max_Payload_Size
0b000	128B
0b001	256B
0b010	512B
0b011	1024B
0b100	2048B
0b101	4096B

“Max_Payload_Size Supported”字段决定了一个 TLP 报文可能使用的最大有效负载，PCIe 总线规定 Max_Payload_Size 参数的最大值为 4096B，但是许多 PCIe 设备并不一定能够支持这么大的有效负载。在实际应用中，一个 PCIe 设备支持的 Max_Payload_Size 参数通常为 128B、256B 或者 512B。

“Max_Payload_Size Supported”字段仅表示该 PCIe 设备允许使用的 Max_Payload_Size 参数。在 Device Control 寄存器中，还有一个 Max_Payload_Size 参数，该字段可以由软件设置，表示实际使用的 Max_Payload_Size 参数大小。

值得注意的是，在 PCIe 设备中，“Max_Payload_Size Supported”参数和 Max_Payload_Size 参数并不相同，前者是一个 PCIe 设备能够支持的最大 Payload 的大小，而后者是链路两端的 PCIe 设备进行协商，确定的实际使用值。有关这两个参数的详细说明见第 6.4 节。

该寄存器的第 4～3 位为 Phantom Functions Supported 字段，该字段只读。当 Device Control 寄存器的 Phantom Functions Enable 位为 1 时，该字段才有意义。

- 该字段为 0b00 时表示不支持 Phantom 功能，PCIe 设备不能使用 Function Number 扩展数据报文的 Tag 字段。
- 该字段为 0b01 时表示支持 Phantom 功能，PCIe 设备可以使用 Function Number 的最高

位扩展 TLP 的 Tag 字段。

- 该字段为 0b10 时表示支持 Phantom 功能，PCIe 设备可以使用 Function Number 的最高两位扩展 TLP 的 Tag 字段。
- 该字段为 0b11 时表示支持 Phantom 功能，PCIe 设备可以使用 Function Number 的全部三位扩展 TLP 的 Tag 字段。

该寄存器的第 5 位为 Extended Tag Field Supported 位，该位为 1 时表示 TLP 的 Tag 字段为 8 位；否则为 5 位。有关 Tag 字段的详细说明见第 6.3.1 节。本节不对该寄存器的其他位进行说明。

3. Device Control 寄存器

该寄存器各字段的描述如表 4-5 所示。

表 4-5　Device Control 寄存器

Bit	定　义	描　述
0	Correctable Error Reporting Enable	该位可读写，其复位值为 0。当此位为 1 时，PCIe 设备可以发出 ERR_COR Messages 报文。而当此位为 0 时，不支持这种操作
1	Non-Fatal Error Reporting Enable	该位可读写，其复位值为 0。当此位为 1 时，PCIe 设备可以发出 ERR_NON-FATAL Messages 报文。而当此位为 0 时，不支持这种操作
2	Fatal Error Reporting Enable	该位可读写，其复位值为 0。当此位为 1 时，PCIe 设备可以发出 ERR_FATAL Messages 报文。而当此位为 0 时，不支持这种操作
3	Unsupported Request Reporting Enable	该位可读写，其复位值为 0。当此位为 1 时，PCIe 设备可以发出 Unsupported Requests Error Messages 报文；而当此位为 0 时，不支持这种操作
4	Enable Relaxed Ordering	该位为 1 时，使能 PCIe 设备的 Relaxed Order 模式，即 PCIe 设备在发送 TLP 时，可以根据需要设置 TLP 的 Attr 字段为 Relaxed Ordering；该位为 0 时，TLP 的 Attr 字段不能设置为 Relaxed Ordering。该位复位时为 1，可读写
7：5	Max_Payload_Size	该字段可读写，PCIe 设备根据 Device Capability 寄存器的 Bit［2：0］字段设置 PCIe 设备 TLP 的最大 Payload。系统软件根据 PCIe 链路两端的实际情况，确认该字段的值。但是该值不能大于 Device Capability 寄存器的“Max_Payload_Size Supported”字段 PCIe 设备发送 TLP 时，其最大 Payload 不能大于 Max_Payload_Size；当 PCIe 设备接收 TLP 时，必须能够处理小于该字段的 TLP，而大于该字段的 TLP 将被认做错误报文
8	Extended Tag Field Enable	该位为 1 时，发送端可以使用 8 位的 Tag 字段；该位为 0 时，可以使用 5 位的 Tag 字段。该字段的复位值为 1，可读写。Tag 字段的详细描述见第 6.3.2 节
9	Phantom Functions Enable	该位为 1，发送端可以使能 Phantom Function 功能；为 0，不使能这个功能。该字段的复位值为 0，可读写。Phantom Function 功能的详细描述见上文
10	Auxiliary（AUX）Power PM Enable	该位为 1 时，PCIe 设备可以使用总线提供的辅助电源
11	Enable No Snoop	此位为 1 时，PCIe 设备在发送 TLP 时，该 TLP 的 Attr 字段可以设置为 No Snoop；该位为 0 时，TLP 的 Attr 字段不能设置为 No Snoop。该位复位时为 1，可读写。该位与 Cache 共享一致性相关
14：12	Max_Read_Request_Size	该字段记录在一个 PCIe 设备中，存储器读请求 TLP 可以请求的最大数据区域。当 PCIe 设备发送存储器读请求 TLP 时，该 TLP 所请求的数据大小不能超过 Max_Read_Request_Size 参数 该字段的关系与表 4-4 中的描述相同

4. Device Status 寄存器

Device Status 寄存器主要字段的含义如表 4-6 所示。

表 4-6　Device Status 寄存器

Bit	定　义	描　述
0	Correctable Error Detected	该位为 1 时表示 PCIe 设备检测到 Correctable Error，对该位写 1 将清除此位
1	Non-Fatal Error Detected	该位为 1 时表示 PCIe 设备检测到 Non-Fatal Error，对该位写 1 将清除此位
2	Fatal Error Detected	该位为 1 时表示 PCIe 设备检测到 Fatal Error，对该位写 1 将清除此位
3	Unsupported Request Detected	该位为 1 时表示 PCIe 设备收到一个 PCIe 总线并不支持的报文请求，对该位写 1 将清除此位
4	AUX Power Detected	当 PCIe 设备检查到辅助电源的存在时，而且如果该设备需要使用辅助电源，则将该位置 1
5	Transactions Pending	对于 EP 而言，该位为 1 表示当前 PCIe 设备发送了一个 Non-Posted 的数据请求，但是没有收到完成报文应答；对于 RC 和 Switch 而言，该位为 1 表示 RC 和 Switch 自身（并不是转发其他设备的 Non-Posted 数据请求）发出了一个 Non-Posted 的数据请求，但是没有收到完成报文应答

5. Link Capabilities 寄存器

Link Capabilities 寄存器描述 PCIe 链路的属性，其主要字段的含义如下。

- Supported Link Speeds 字段。为 0b0001 表示 PCIe 链路支持 2.5GT（gigatransfers）/s；为 0b0010 表示 PCIe 链路支持 5GT/s；为 0b0100 表示 PCIe 链路支持 8 GT/s。
- Maximum Link Width 字段。该字段存放该 PCIe 设备支持的最大链路宽度。该字段为 0b000001 表示最大支持 ×1 的 PCIe 链路；为 0b000010 表示最大支持 ×2 的 PCIe 链路；为 0b000100 表示最大支持 ×4 的 PCIe 链路；为 0b001000 表示最大支持 ×8 的 PCIe 链路；为 0b001100 表示最大支持 ×12 的 PCIe 链路；为 0b010000 表示最大支持 ×16 的 PCIe 链路；为 0b100000 表示最大支持 ×32 的 PCIe 链路。
- ASPM（Active State Power Management）Support 字段，该字段只读。0b00 和 0b10 为系统保留字段。当该字段为 0b01 时，表示 ASPM 支持 L0s 状态；当该字段为 0b11 时，表示 ASPM 支持 L0s 和 L1 状态。PCIe 设备除了支持 PCI PM 电源管理方式之外，还支持 ASPM 机制进行电源管理。ASPM 机制是 PCIe 设备进行的主动电源管理方式，与系统软件没有直接联系。有关 ASPM 的详细描述见第 8.3 节。
- L0s Exit Latency 和 L1 Exit Latency 字段。这两个字段定义了 PCIe 设备从 L0s 和 L1 状态退出的最小延时。
- Port Number 字段。如果多端口 RC 和 Switch 支持多个下游端口，则使用该字段对这些端口进行编号。PCIe 设备进行链路训练时，需要使用这个端口号。

6. Link Control 寄存器

Link Control 寄存器主要字段的解释如下。

- ASPM Control 字段，该字段可读写。该字段为 0b00 时表示禁止 PCIe 设备的 ASPM 机制；为 0b01 时表示 PCIe 设备可以进入 L0s 状态；为 0b10 时表示 PCIe 设备可以进入 L1 状态；为 0b11 时表示 PCIe 设备可以进入 L0s 和 L1 状态。值得注意的是系统软件不能通过修改该字段使 PCIe 链路进入相应的状态，仅是通知硬件逻辑，可以进入相应的状态。ASPM 的详细描述见第 8.3 节。

- RCB((Read Completion Boundary))位。该位为0时，表示RCB为64B，该位为1时，表示RCB为128B。RCB的大小与完成报文的有效负载相关。对于RC而言，该字段只读，而Switch和EP可以读写该字段。有关该位的进一步说明见第6.4.3节。
- Link Disable位。向此位写1，将禁止PCIe链路。此时链路状态机将进入Disabled状态，有关PCIe链路状态机的详细说明见第8.2节。
- Retrain Link位。向此位写1，将重新训练PCIe链路。此时PCIe链路状态机将进入Recovery状态。
- Common Clock Configuration位，该位可读写。当该位为1时，表示PCIe链路两端的设备使用同源的参考时钟，即相同的REFCLK差分时钟；如果该位为0，表示PCIe链路两端的设备使用的参考时钟并不同源，即使用异步时钟。
- Extended Sync位，该位可读写。当该位为1时，表示PCIe设备退出L0s和进入Recovery状态时，需要额外发出一些同步序列。
- Hardware Autonomous Width Disable位，该位可读写。当该位为1时，PCIe设备不能改变当前已经协商好的PCIe链路宽度，除非为了修正PCIe链路中已经出现错误的Lane。
- Link Bandwidth Management Interrupt Enable位，该位可读写。当该位为1时，且Link Status寄存器的Link Bandwidth Management Status位为1时，PCIe设备将向处理器提交中断请求。此时这个中断请求使用的中断向量由PCI Express Capability寄存器的Interrupt Message Number字段确定。
- Link Autonomous Bandwidth Interrupt Enable位，该位可读写。当该位为1时，且Link Status寄存器的Link Autonomous Bandwidth Status位为1时，PCIe设备将向处理器提交中断请求。

7. Link Status寄存器

Link Status寄存器存放PCIe设备正在使用的PCIe链路的状态，由链路宽度和速度等参数组成，其主要字段的含义如表4-7所示。

表4-7　Link Status寄存器

Bit	定　义	描　述
3：0	Current Link Speeds	为0b0001表示PCIe链路的传输率为2.5GT/s；为0b0010表示PCIe链路的传输率为5GT/s；为0b0100表示PCIe链路的传输率为8GT/s。该字段只读
9：4	Negotiated Link Width	该字段存放当前PCIe设备和其上游PCIe设备进行链路协商后使用的链路宽度。该字段为0b000001表示使用×1的PCIe链路；为0b000010表示使用×2的PCIe链路；为0b000100表示使用×4的PCIe链路；为0b001000表示使用×8的PCIe链路；为0b001100表示使用×12的PCIe链路；为0b010000表示使用×16的PCIe链路；为0b100000表示使用×32的PCIe链路。该字段在PCIe链路进行训练的过程中，由硬件逻辑写入，系统软件只能读取该字段
11	Link Training	该位只读，为1时，表示PCIe链路正处于重新配置和重新训练阶段，当PCIe链路结束上述操作时，将此位清零
12	Slot Clock Configuration	该位由PCIe设备在初始化时确定，该位为1表示PCIe插槽与Add-In卡使用的参考时钟源相同。读者需要留意该位与Common Clock Configuration位的差别
13	Data Link Layer Link Active	该位表示PCIe链路的状态。该位为1时，表示PCIe链路处于DL_Active，即正常工作状态

（续）

Bit	定　义	描　述
14	Link Bandwidth Management Status	该位由 PCIe 硬件设置。当 PCIe 链路重训练结束，或者 PCIe 设备完成 PCIe 链路的链路宽度和链路速度的设定后，该位置 1。该位写 1 清除
15	Link Autonomous Bandwidth Status	该位由 PCIe 设备确定。当 PCIe 链路自主完成链路宽度和速度的协商后，将该位置 1。该位写 1 清除

8. Device Capabilities 2 寄存器

该寄存器定义了一些 PCIe V2.1 总线规范使用的字段，其主要字段如表 4-8 所示。

表 4-8　Device Capabilities 2 寄存器

Bit	定　义	描　述
6	AtomicOp Routing Supported	Switch 的上下游端口和 RC 端口支持该位，在 PCIe V2.1 总线规范定义了原子操作。当该位为 1 时，表示原子操作 TLP 可以通过当前 Switch 或者 RC。有关原子操作的详细描述见第 6.3.5 节
7	32-bit AtomicOp Completer Supported	该位为 1 时，表示 EP 或者 RC 支持 32 位原子操作
8	64-bit AtomicOp Completer Supported	该位为 1 时，表示 EP 或者 RC 支持 64 位原子操作
9	128 - bit CAS Completer Supported	该位为 1 时，表示 EP 或者 RC 支持 128 位原子操作。在 PCIe 总线规范中，只有 CAS（Compare and Swap）支持 128 位操作
13；12	TPH Completer Supported	该字段为 0b00 时，表示接收端不支持 TPH 和扩展 TPH 报文；为 0b01 时，表示接收端仅支持 TPH 报文；为 0b11 时，表示接收端支持 TPH 和扩展 TPH 报文；而 0b10 保留。有关 TPH 的详细描述见第 6.3.6 节
20	Extended Fmt Field Supported	该位为 1 时，表示 TLP 的 Fmt 字段为 3 位，即支持 TLP Prefix；为 0 时，Fmt 字段为 2 位。Fmt 字段的详细描述见第 6.1.1 节。该位由 V2.1 规范引入，其目的是为了扩展 TLP 头
21	End-End TLP Prefix Supported	该位为 1 时，表示 EP 可以接收含有 End-End TLP Prefix 的 TLP
23～22	Max End-End TLP Prefixes	该字段限制一个 TLP 中 End-End TLP Prefix 的个数。该字段为 0b01 表示 TLP 中最多含有 1 个 End-End TLP Prefix；为 0b10 表示最多含有 2 个 End-End TLP Prefix；为 0b11 表示最多含有 3 个 End-End TLP Prefix；为 0b00 表示最多含有 4 个 End-End TLP Prefix

9. Device Control 2 寄存器

Device Control 2 寄存器主要字段的含义如表 4-9 所示。

表 4-9　Device Control 2 寄存器

Bit	定　义	描　述
6	AtomicOp Requester Enable	该位可读写，对 EP 和 RC 有效。如果该位和 Command 寄存器的“Bus Master Enable”位同时有效，EP 或者 RC 可以发出原子操作请求 TLP
7	AtomicOp Egress Blocking	该位可读写，对 Switch 的上下游端口和 RC 端口有效。当 AtomicOp Routing Supported 位为 1 时，该位可以为 1，否则该位只能为 0。该位为 1 时，Egress 端口将阻止原子操作 TLP 通过
8	IDO Request Enable	该位可读写，对 EP 和 RC 有效。当该位为 1 时，TLP 中的 IDO（ID-Based Ordering）位可以根据实际情况（即 TLP 的 Attr2 位）设置为 1。IDO 是 PCIe V2.1 总线规范引入的新的“序”模型。有关 IDO 机制的详细说明见第 6.1.3 节

（续）

Bit	定　义	描　述
9	IDO Completion Enable	该位可读写。当该位为 1 时，EP 可以处理 IDO 位为 1 的完成报文
15	End-End TLP Prefix Blocking	该位可读写。当该位为 0 时，EP 不能发送带有 End-End TLP Prefix 的 TLP；为 1 时，可以发送

10. Root Control 和 Root Status 寄存器

这两个寄存器与 PCIe 总线的 AER（Advanced Error Reporting）机制相关。其中 Root Control 寄存器由以下位组成。

- System Error on Correctable Error Enable。该位为 1 时，表示 RC 端口管理的 PCI 树或者 RC 端口发送 ERR_COR 信息后，将向处理器提交 System Error 信息。
- System Error on Non-Fatal Error Enable。该位为 1 时表示 RC 端口管理的 PCI 树或者 RC 端口发送 ERR_NONFATAL 信息后，将向处理器提交 System Error 信息。
- PME Interrupt Enable。该位为 1 时，如果 RC 端口收到 Root Status 寄存器的 PME Status 为 1 的信息后，将向处理器提交 PME 中断信息。
- CRS Software Visibility Enable。当此位为 1 时，系统软件发送配置请求 TLP 后，RC 端口可以要求该配置请求 TLP 择时重试。如当 PCIe 总线没有初始化完毕时，不能接收处理器的配置请求，此时将该位置 1；初始化完毕后，将该位置 0。

而 Root Status 寄存器由以下位和字段组成。

- PME Requester ID。该字段记录最后发送 PME 消息的 PCIe 设备的 Requester ID 号。
- PME Status。该位为 1 时，表示 PCIe 设备（ID 号为 PME Requester ID）已经向 RC 发送了 PME 消息，但是并没有被处理完毕，该位写 1 清除。
- PME Pending。当 PME Status 位为 1 而且该位也 1 时，表示 RC 中有尚未处理的 PME 消息。当 RC 清除 PME Status 位后，硬件将向 RC 提交 PME 消息，更新 PME Requester ID，并将 PME Status 重新置为 1，同时清除 PME Pending。PCIe 总线设置该位的主要目的是为了防止丢失 PME 消息。

PCI Express Capability 结构中还含有 Slot Capabilities、Slot Control 和 Slot Status 等寄存器。为节约篇幅，本节并不对这些寄存器一一进行介绍，在一个指定的 PCIe 设备中，PCI Express Capability 结构的这些寄存器并不会全部实现。许多 PCIe 设备甚至不存在 PCI Express Capability 结构，但是在这些 PCIe 设备中依然存在与 PCI Express Capability 结构相关的概念，只是这些结构没有以寄存器的形式表现出来，供系统程序员使用而已。

4.3.3　PCI Express Extended Capabilities 结构

PCI Express Extended Capabilities 结构存放在 PCI 配置空间 0x100 之后的位置，该结构是 PCIe 设备独有的，PCI 设备并不支持该结构。实际上绝大多数 PCIe 设备也并不支持该结构。在一个 PCIe 设备中可能含有多个 PCI Express Extended Capabilities 结构，并形成一个单向链表，其中第一个 Capability 结构的基地址为 0x100，其结构如图 4-19 所示。

在这个单向链表的尾部，其 Next Capability Offset、Capability ID 和 Capbility Version 字段的值都为 0。如果在 PCIe 设备中不含有 PCI Express Extended Capabilities 结构，则 0x100 指

针所指向的结构，其 Capability ID 字段为 0xFFFF，而 Next Capability Offset 字段为 0x0。

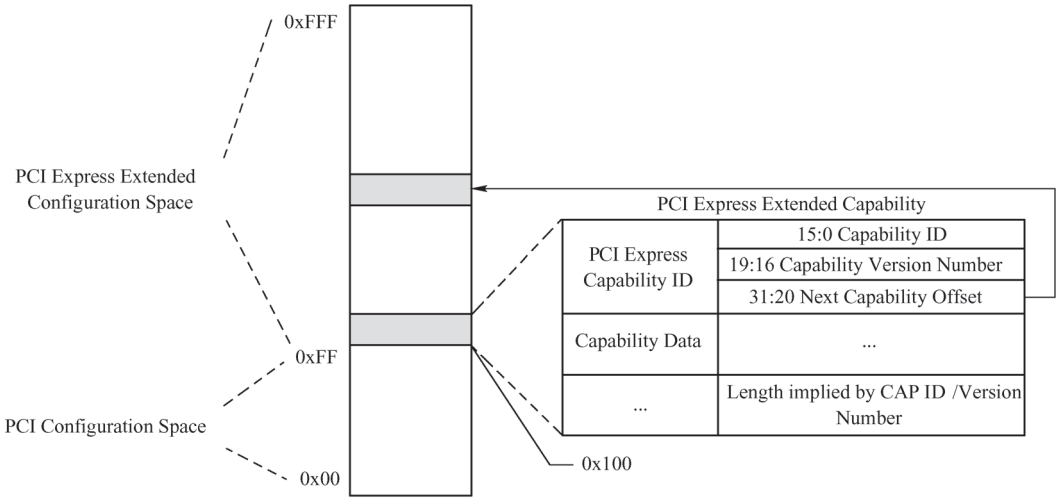

图 4-19　PCI Express Extended Capabilities 结构

一个 PCI Express Extended Capabilities 结构由以下参数组成。

- PCI Express Capability ID 字段存放 Extended Capability 结构的 ID 号。
- Capability Version 字段存放 Extended Capability 结构的版本号。
- Next Capability Offset 字段存放下一个 Extended Capability 结构的偏移。

PCIe 总线定义了一系列 PCI Express Extended Capabilities 结构，如下所示。

- AER Capability 结构。该结构定义了所有 PCIe 设备可能遇到的错误，包括 Uncorrectable Error（不可恢复错误）和 Correctable Error（可恢复错误）。当 PCIe 设备发现这些错误时，可以根据该寄存器的设置使用 Error Message 将错误状态发送给 Event Collector，并由 Event Collector 统一处理这些错误。系统软件必须认真处理每一个 Error Message，并进行恢复。对一个实际的工程项目，错误处理是保证整个项目可靠性的重要一环，不可忽视。AER 机制与 Error Message 报文的处理相关，第 6.3.4 节将进一步介绍 AER 机制。
- Device Serial Number Capability 结构。该结构记载 PCIe 设备使用的序列号。IEEE 定义了一个 64 位宽度的 PCIe 序列号，其中前 24 位作为 PCIe 设备提供商使用的序列号，而后 40 位由厂商选择使用。
- PCIe RC Link Declaration Capability 结构。在 RC、RC 内部集成的设备或者 RCRB 中可以包含该结构。该结构存放 RC 的拓扑结构，如 RC 使用的 PCI 链路宽度。如果 RC 支持多个 PCIe 链路，该结构还包含每一个链路的描述和端口命名。
- PCIe RC Internal Link Control Capability 结构。该结构的主要作用是描述 RC 内部互连使用的 PCIe 链路。该结构由 Root Complex Link Status 和 Root Complex Link Control 寄存器组成。
- Power Budget Capability 结构。当处理器系统为一些动态加入的 PCIe 设备分配电源配额时，将使用该设备的 Power Budget Capability 结构。
- ACS（Access Control Services）Capability 结构。该结构对 PCIe 设备进行访问控制管

理。RC 端口、Switch 的下游端口和多功能 PCIe 设备可以支持该结构。该结构与 PCIe 总线的 ACS 机制相关。ACS 机制定义了一组与收到的 TLP 相关的操作，该机制的原理较为简单，本节对此不做进一步分析。

- RCRB Header Capability 结构。该结构存放 RC 中的 RCRB，第 5.1 节将以 Montivina 平台为例介绍该结构的组成结构。
- RC Event Collector EP Association Extended Capability。在 x86 处理器系统中，RC 包含一个 Event Collector 控制器，该控制器处理 PCIe 设备发向 RC 的各类消息，如 PME 消息和 Error 消息。该结构用来描述 Event Collector 控制器。
- Multicast Capability 结构。PCIe 总线上的 RC、Switch 或者 EP 如果支持 Multicast 消息，需要使用该结构描述支持哪些 Multicast 组。PCIe 体系结构支持 Multicast 功能，在 PCIe 总线中，除了 PCIe 桥一定不支持 Multicast 功能外，其他设备都可以支持该功能。本节对 Multicast 功能不做进一步说明。

此外 PCIe 总线还可以支持其他 Capability 结构，如 Vendor-Defined Capability、Resizable BAR Capability、DPA（Dynamic Power Allocate）和 MFVC（Multi-Function Virtual Channel）Capability 结构等其他结构。但是在 PCIe 总线中，这些扩展的 Capability 结构并没有得到充分利用。在一个实际的 PCIe 设备中可能并不包含这些结构。

PCIe 设备定义的 Capability 结构有些过多，使用这种方法可以概括所有 PCIe 设备的使用特性。许多 PCIe 设备在支持这些 Capability 结构后，几乎可以不使用 BAR 寄存器空间存放与 PCIe 总线相关的任何信息。但是过多的 Capability 结构为软硬件工程师在设计上带来了不小的麻烦。一般说来，事务的发展过程是由简入繁，由繁化简。目前 PCIe 总线的发展仍处在由简入繁的过程。

本节仅详细介绍 PCI Express Extended Capabilities 结构组中的 MFVC 结构。MFVC 结构是 PCIe 总线的一个可选结构，其结构如图 4-20 所示。TLP 在通过 Switch 时需要通过 TC/VC Mapping，而且在进行 VC 仲裁和端口仲裁时，需要使用某些仲裁策略。

在 PCIe 总线中，TC/VC Mapping 表和 VC/端口仲裁策略在 MFVC 结构中定义。其结构如图 4-20 所示。值得注意的是，在许多 PCIe 设备中，可能只具有一个 VC，而且其 VC 仲裁的算法固定，那么在这个 PCIe 设备中，MFVC 结构并没有存在的必要。目前支持多 VC 的 PCIe 设备极少，仅有一些 RC 和 Switch 中存在多个 VC，而且也仅支持两个 VC。

VC Capability 的 ID 为 0x02 或者 0x09，VC Capability 结构由两部分组成，分别是一个 VC Capability 寄存器组和 n 个 VC Resource 寄存器组，其中 VC Resource 寄存器是可选的。如果 PCIe 设备仅支持一个 VC 时，该结构中不含有 VC Resource 寄存器，而在 VC Capability 寄存器组中包含该 VC 的描述信息。当一个 PCIe 设备支持 8 个 Function 时，则 n 为 7；如果支持 7 个设备，则 n 为 6，并以此类推。其中每一个 VC Resource 寄存器组中都包含一个 VC 仲裁表、端口仲裁表和 VC/TC 的映射表。

1. VC Capability 寄存器组

该组寄存器由 Port VC Capability Register 1、Port VC Capability Register 2、Port VC Control Register 和 Port VC Status Register 寄存器组成。

（1）Port VC Capability Register 1 主要字段的含义如表 4-10 所示。

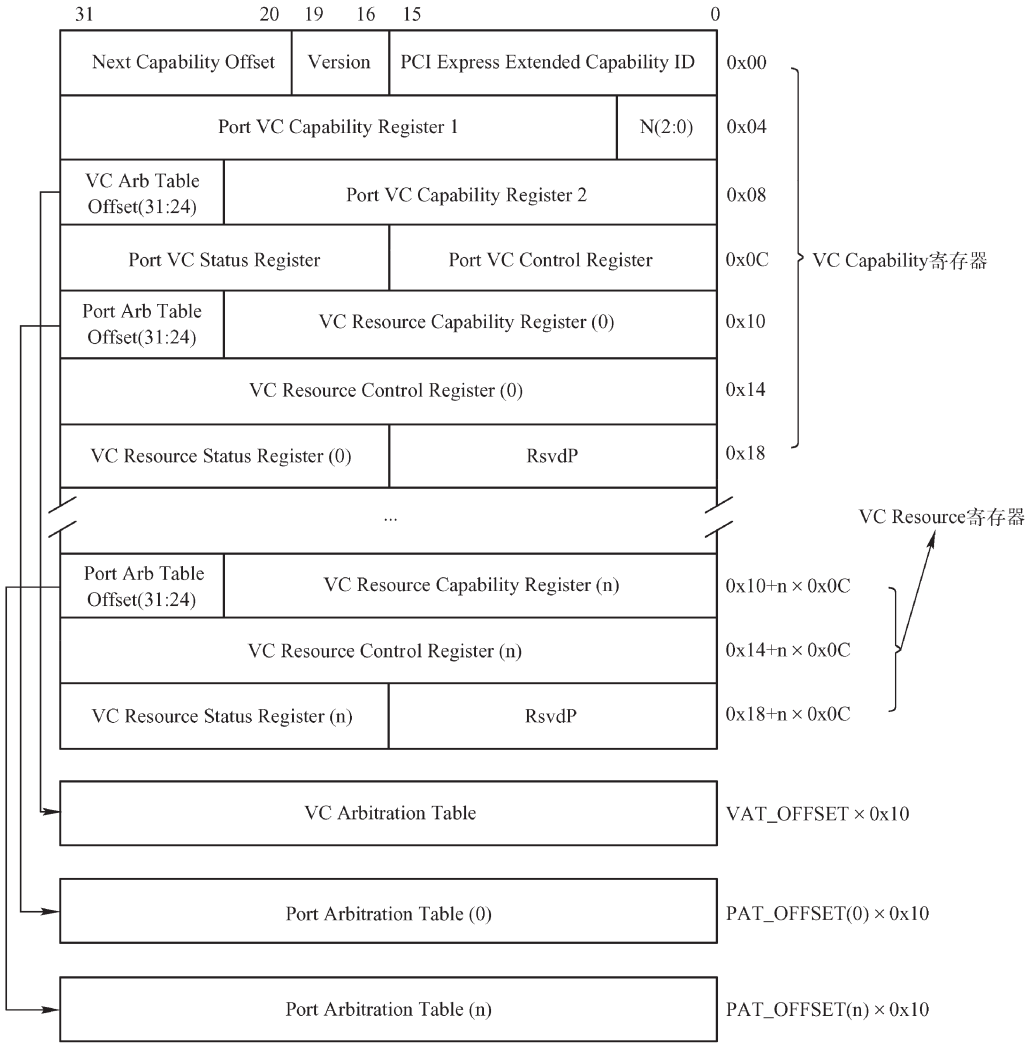

图 4-20　MFVC 结构

表 4-10　Port VC Capability Register 1

Bit	定　义	描　述
2：0	Extended VC Count	扩展的 VC 个数，最小值为 0，表示只支持 VC0；最大值为 7，表示支持 8 个 VC，VC0 ~ VC7
6：4	Low Priority Extended VC Count	和 VC0 优先级相同的扩展 VC 的个数。在 PCIe 总线中，VC0 的级别最低。该字段的最小值为 0，最大值为 7
9：8	Reference Clock	如果 VC 使用 Time-based WRR 算法时，需要使用一个参考时钟。PCIe 总线规定当该字段为 0b00 时，这个参考时钟的周期为 100 ns，即时钟频率为 10 MHz
11：10	Port Arbitration Table Entry Size	表示 Port Arbitration Table Entry 的大小。0b00 表示 Entry 的长度为 1 位；0b01 表示 Entry 的长度为 2 位；0b10 表示 Entry 的长度为 4 位；0b11 表示 Entry 的长度为 8 位。

（2）Port VC Capability Register 2

该寄存器由两个字段组成。其中 VC Arbitration Capability 字段存放 PCIe 设备支持的 VC 调度算法，PCIe 总线提供的调度算法包括 Hardware-fixed 仲裁策略和 WRR 仲裁策略。其中

WRR 仲裁策略分为 32、64 或者 128 个 Phase，VC 仲裁不支持 Time-based WRR 算法，有关 WRR 算法的详细说明见下文。

而 VC Arbitration Table Offset 字段存放 VC Arbitration Table 的地址偏移，如果在 PCIe 设备中不含有 VC Arbitration Table，该字段为 0。

（3）Port VC Control Register

该寄存器由 Load VC Arbitration Table 位和 VC Arbitration Select 字段组成。系统软件通过操纵该寄存器更改 VC 的仲裁算法，其中 VC Arbitration Select 字段用来选择 VC Arbitration Table 的长度，其关系如表 4-11 所示。

表 4-11　VC Arbitration Select 字段的说明

VC Arbitration Select 字段	VC Arbitration Table 的长度
0b001	32
0b010	64
0b011	128

Load VC Arbitration Table 位用来更新 VC 的仲裁算法。系统软件向 Load VC Arbitration Table 位写 1 时更新 VC 的仲裁算法，PCIe 设备可以根据 VC Arbitration Select 字段选择合适的仲裁算法，系统软件向 Load VC Arbitration Table 位写 0 没有意义。

（4）Port VC Status Register

Port VC Status Register 使用 VC Arbitration Table Status 位，控制更新 VC 仲裁算法的进度。当系统软件向 Load VC Arbitration Table 位写 1 时，PCIe 设备将更新 VC 的仲裁算法，并将 VC Arbitration Table Status 位置 1。PCIe 设备在没有完成仲裁算法的更换之前，VC Arbitration Table Status 位一直为 1，当 PCIe 设备完成仲裁算法的更换后，该位被 PCIe 设备清零。

2. VC Resource 寄存器组

在 PCIe 设备中，每一个 VC 都有一组 VC Resource 寄存器组，这组寄存器设置每一个 VC 的属性和端口仲裁算法。该组寄存器由 VC Resource Capability Register、VC Resource Control Register 和 VC Resource Status Register 寄存器组成。

（1）VC Resource Capability Register 主要字段的含义如表 4-12 所示。

表 4-12　VC Resource Capability Register

Bit	定　义	描　述
7：0	Port Arbitration Capability	存放当前 VC 支持的端口仲裁算法。该字段对 Switch 和支持 Peer-to-Peer 传送的 RC 端口有效 Bit 0：硬件固化的算法，如 RR Bit 1：WRR with 32 phases Bit 2：WRR with 64 phases Bit 3：WRR with 128 phases Bit 4：Time-based WRR with 128 phases Bit 5：WRR with 256 phases Bit 6～7：保留
15	Reject Snoop Transactions	此位为 0 时，TLP 的 No Snoop 位无论是 0 还是 1 都可以通过该 VC；此位为 1 时，如果 TLP 的 No Snoop 位为 0，该 TLP 不能通过该 VC。该位对 RC 或者 RCRB 有意义

（2）VC Resource Control Register 主要字段的含义如下。

- TC/VC Map 字段，第 7～0 位。该字段的每一位对应一个 TC，其中第 7 位对应 TC7，

该位有效时表示 TC7 使用该 VC 进行数据传递；第 6 位对应 TC6，该位有效时表示 TC6 使用该 VC 进行数据传递，并以此类推。对于 VC0 通路，该字段的复位值为 0xFF，对于其他 VC 通路，该字段的复位值为 0x00。因此在系统初始化时，所有 TC 都使用 VC0 进行数据传递，而 PCIe 链路必须支持 VC0。使用该字段可以保证 TC 不同的 TLP 可以使用同一个 VC，但是在 PCIe 总线中，一个 TC 与一条 VC 建立了映射关系后，不能与其他 VC 建立映射关系。

- Load Port Arbitration Table 位，第 16 位。当该位被置 1 后，PCIe 设备将使用 Port Arbitration Table 更新端口仲裁的算法，当该位置 1 后，VC Resource Status 寄存器的 Port Arbitration Table Status 位也将置 1，当端口仲裁算法更新完毕后，Port Arbitration Table Status 位将清零；对此位写 1 没有意义。
- Port Arbitration Select 字段，第 19 ~ 17 位。该字段描述 Port Arbitration Table 表的 Entry 个数。下文将详细解释该字段。
- VC ID 字段，第 26 ~ 24 位。该字段存放当前 VC 的 ID 号，PCIe 设备的第一个 VC ID 必须为 0。
- VC Enbale 位，第 31 位。该位为 1 时，当前 VC 通路有效，否则无效。在系统初始化完毕后，VC0 的 VC Enable 位为 1，而其他 VC 的 VC Enable 位为 0。

（3）VC Resource Status Register 主要字段的含义如下。

- Port Arbitration Table Status 位，第 0 位。该位表示当前 VC 更新端口仲裁算法的状态，该位由 PCIe 设备维护，对系统软件只读。
- VC Negotiation Pending 位，第 1 位。该位为 1 表示当前 VC 通路正在进行初始化或者处于正在关闭的状态，此时当前 VC 并没有准备好，还没有从 FC_INIT2 状态中退出；为 0 表示当前 VC 准备好，PCIe 链路已经完成流量控制的初始化。系统软件必须保证该位为 0 后，才能对该 VC 进行操作。有关 FC_INIT2 状态的详细说明见第 9.3.3 节。

3. VC Arbitration Table

VC Arbitration Table 的长度由 Port VC Control 寄存器的 VC Arbitration Select 字段确定，最小为 32 个 Entry，最大为 128 个 Entry。VC Arbitration Table 实现 VC 仲裁的 WRR 算法。在 VC Arbitration Table 中，每一个 Entry 由 4 位组成，其中最高位保留，最低三位记录 VC 号。下文举例说明 32 个 Phase 的 WRR 算法，在这种情况下 VC Arbitration Table 的长度为 32，这个表中每一个 Entry 记录一个 VC 号，如图 4-21 所示。

在图 4-21 中，VC Arbitration Table 的每一个 Entry 都记录一个 VC 号。假定 VC 仲裁时从 Phase0 开始使用，该 Entry 存放的 VC 号为 VC0，则 VC 仲裁的结果是传送虚通路 VC0 中的总线事务，当这个总线事务传送结束后，将处理 Phase1 中的 VC；如果该 Entry 存放的 VC 号为 VC2，则 VC 仲裁的结果是传送虚通路 VC2 中的总线事务，并以此类推直到 Phase31 后，再对 Phase0 重新进行处理。

使用这种加权处理的方法，可以保证 PCIe 总线 QoS。值得注意的是，使用该方法时，如果当前 Entry 存放的 VC 中，不存在总线事务时，将迅速移动到下一个 Entry；如果 VC 间并没有出现冲突时，不需要使用该表进行仲裁，使用 64 个 Phase 和 128 个 Phase 的 WRR 算法的实现机制与此类似。由上文的分析可以发现，与 RR 算法相比，PCIe 总线使用 WRR 算法可以在保证 QoS 的基础上，使各个 VC 公平使用端口资源。

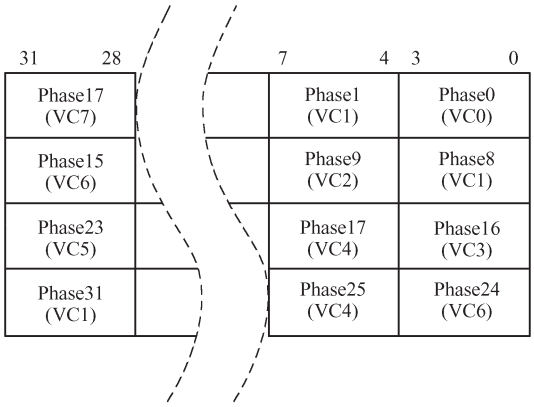

图 4-21　32 Phases 的 VC Arbitration Table

4. Port Arbitration Table

每一个 VC 都有一个 Port Arbitration Table，如图 4-12 所示。每一个 TLP 都首先需要进行端口仲裁之后，才能进行 VC 仲裁，然后通过端口发送。Port Arbitration Table 的主要作用是确定端口仲裁的策略。其长度由 VC Resource Capability Register 的 Port Arbitration Capability 字段确认，如表 4-12 所示。

在该表中，每一个 Entry 的大小由该设备支持的端口数目有关，如果一个设备支持 N 个端口，则该表 Entry 的大小为 $\lceil \mathrm{Log_2 N} \rceil$。如果一个设备有 6 个端口，则 Port Arbitration Table 的 Entry 大小为 3。PCIe 总线支持 RR、WRR 和 Time-based WRR 端口仲裁策略。

Time-based WRR 端口仲裁策略的引入是为了支持 PCIe 总线的 isochronous 数据传送方式。在 PCIe 总线中使用 WRR 算法每处理完一个总线事务将移动一个 Phase，而 Time-based WRR 算法需要至少经过一个时间槽后才能移动一个 Phase。PCIe 总线为 Time-based WRR 算法使用的基准时钟周期在 Port VC Capability Register 1 的 Reference Clock 字段中定义，目前该值为 100ns。

PCIe 总线中使用的这些仲裁算法源于网络通信，这几种算法都是基于轮询的仲裁算法。在网络中，还经常使用 DWRR（Deficit Weighted Round Robin）算法。

WRR 算法在支持长度不同的报文时，会出现带宽分配不公平的现象，为此 M. Shreedhar 与 George Varghese 提出了 DWRR 调度算法。DWRR 算法给每一个队列分配的权值不是基于报文的个数，而是基于报文的比特数。因此可以使各个队列公平地获得带宽。但是这种算法并不适用于 PCIe 总线，因为 PCIe 总线基于报文进行数据传递，而不是基于数据流。该算法在 ATM 分组交换网中得到了广泛的应用。

4.4　小结

本章简要介绍了 PCIe 总线的各个组成部件，包括 RC、Switch 和 EP 等，并介绍了 PCIe 总线的层次组成结构，和 PCIe 设备使用的 Capability 结构。本章是读者了解 PCIe 体系结构的基础。

第 5 章　Montevina 的 MCH 和 ICH

本章以 Montevina 平台为例，说明在 x86 处理器系统中，PCIe 体系结构的实现机制。Montevina 平台是 Intel 提供的一个笔记本平台。在这个平台中，含有一个 Mobile 芯片组、Mobile 处理器和无线网卡。其中 Mobile 芯片组包括代号为“Contiga”的 GMCH（Graphics and Memory Controller Hub）和 ICH9M 系列的 ICH；Mobile 处理器使用代号为“Penryn”的第二代 Intel Core2 Duo；无线网卡的代号为“Shirley Peak”（支持 WiFi）或者“Echo Peak”（同时支持 WiFi 和 WiMax）。Montevina 平台的拓扑结构如图 5-1 所示。

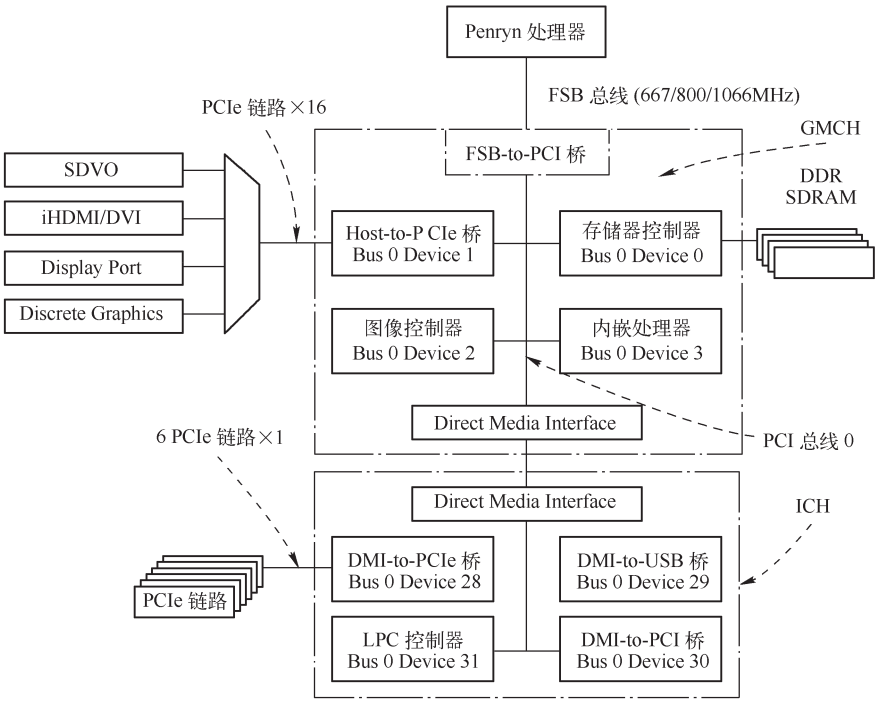

图 5-1　Montevina 平台的拓扑结构

Montevina 平台使用一个虚拟的 FSB-to-PCI 桥[⊖]将 FSB 总线与外部设备分离，这个虚拟 PCI 桥的上方连接 FSB 总线，之下连接 PCI 总线 0。但是从物理信号的角度来看，MCH 中的 PCI 总线 0 是 FSB 总线的延伸，因为该 PCI 总线 0 依然使用 FSB 总线的信号，只是连接到这条总线上的设备相当于虚拟 PCI 设备。在 GMCH 中，并没有提及这个 FSB-to-PCI 桥，但是在芯片设计中，存在这个桥片的概念。

从系统软件的角度来看，在 PCI 总线 0 上挂接的设备都含有 PCI 配置寄存器，系统软件将这些设备看做 PCI 设备，并可以访问这些设备的 PCI 配置空间。在 Montevina 平台的

⊖ 在 Montevina 平台的数据手册中并没有提及这个 FSB-to-PCI 桥。

GMCH 和 ICH 中，所有的外部设备，如存储器控制器，图形控制器等都是虚拟 PCI 设备，都具有独立的 PCI 配置空间。GMCH 和 ICH 之间使用 DMI（Direct Management Interface）接口相连，但是 DMI 接口仅仅是链路级别的连接，并不产生新的 PCI 总线号，ICH 的 DMI-to-USB 桥和 DMI-to-PCIe 桥也都属于 PCI 总线 0 上的设备。

在 x86 处理器中，MCH 包含的虚拟 PCI 设备优先级较高，而 ICH 包含的虚拟 PCI 设备优先级较低。当 CPU 发起一个 PCI 数据请求时，MCH 的 PCI 设备将首先在 PCI 总线 0 上进行正向译码。如果当前 PCI 数据请求所使用的地址没有在 MCH 的 PCI 设备命中时，DMI 接口部件将使用负向译码方式被动地接收这个数据请求，然后通过 DMI 总线将这个数据请求转发到 ICH 中。

因此在 x86 处理器中，MCH[⊖]集成了一些对带宽要求较高的虚拟 PCI 设备，如 DDR 控制器、显卡等。而在 ICH 中集成了一些低速 PCIe 端口，和一些速度相对较低的外部设备，如 PCI-to-USB 桥、LPC 总线控制器等。

MCH 和 ICH 包含一些内置的 PCI 设备，这些设备都具有 PCI 配置空间，x86 处理器可以使用 PCI 配置周期访问这些 PCI 配置空间。在 MCH 和 ICH 中，PCI 总线 0 是 FSB 总线的延伸，所以处理器访问这些设备时并不使用 PCI 总线规定的信号，如 FRAME#、TRDY#、IRDY#和 IDSEL 信号。在 MCH 和 ICH 中，有些 PCI 设备并不是传统意义上的外部设备，而仅是虚拟 PCI 设备，即使用 PCI 总线的管理方法统一在一起的设备。

x86 处理器使用这些虚拟 PCI 外设的优点是可以将所有外部设备都用 PCI 总线统一起来，这些设备使用的寄存器都可以保存在 PCI 设备的配置空间中，但是使用这种方法在某种程度上容易混淆一些概念，尤其是有关地址空间的概念。例如在处理器体系结构的典型定义中，DDR-SDRAM 空间属于存储器域，与其相关的 DDR-SDRAM 控制器也应该属于存储器域，但是在 x86 处理器中存储器控制器属于 PCI 总线域。

5.1 PCI 总线 0 的 Device 0 设备

PCI 总线 0 上存储器控制器（Device 0）是一个比较特殊的 PCI 设备，这个设备除了需要管理 DDR SDRAM 之外，还管理整个存储器域的地址空间，包括 PCI 总线域地址空间。在 x86 处理器系统中，该设备是管理存储器域空间的重要设备，其中含有许多与存储器空间相关的寄存器。

这些寄存器对于系统程序员理解 x86 处理器的存储器拓扑结构非常重要，对底层编程有兴趣的系统程序员需要掌握这些寄存器。但是在 x86 处理器系统中，由于 BIOS 的存在，绝大多数系统程序员并没有机会实际使用这些寄存器。

从底层开发的角度上看，x86 处理器系统并不如 PowerPC、MIPS 和 ARM 处理器透明。x86 处理器首先使用 BIOS 屏蔽了处理器的硬件实现细节，其次在处理器内核中使用了 Microcode[⊖]进一步屏蔽了 CPU 的实现细节。这使得底层程序员在没有得到充分的资源时，几乎无法开发 x86 处理器的底层代码。

⊖ 从体系结构的角度上看，MCH 和 ICH 仅仅是一个称呼，实际上并不重要。

⊖ 许多处理器和外部设备都使用了 Microcode 屏蔽 CPU 的实现细节，如 Alpha 处理器的 PAL。

但是不可否认的是 x86 处理器底层开发的复杂程度超过 PowerPC、MIPS 和 ARM 处理器，因为 x86 处理器系统作为通用 CPU 需要与各类操作系统兼容，而向前兼容对于任何一种处理器都是一个巨大的包袱。x86 处理器系统使用 BIOS 和 Microcode 屏蔽硬件细节基于许多深层次的考虑，包括技术和商业上的考虑，这种做法在 PC 领域取得了巨大的成功。

从传统外部设备的角度上看，PCI 总线 0 的 Device 0 并不是一个设备，仅存放与处理器系统密切相关的一组参数。而除了 x86 处理器之外，几乎所有处理器都使用存储器映射寻址的寄存器保存这些参数。

x86 处理器需要考虑向前兼容，因此存在许多独特的设计。这些独特的设计极易使一些初学者混淆计算机体系结构中的一些基本概念。从这个角度来看，x86 处理器并不适合教学，但这并不影响 x86 处理器在 PC 领域的地位。值得注意的是，在 x86 处理器中，PCI 总线 0 的 Device 0 的存在并不完全是为了向前兼容，而是 Intel 使用 PCI 总线概念统一所有外部设备的方法。

在 Montevina 平台中，系统软件使用 Type 00h 配置请求访问存储器控制器，该存储器控制器除了具有一个标准 PCI Agent 设备的 64B 的配置空间之外，还使用了 PCI 设备的扩展配置空间，其包含的主要寄存器如表 5-1 所示。

表 5-1　Device 0 的基本配置空间

寄存器名	简　写	缺省值	说　明
Vendor Identification	VID	0x8086	Intel 公司使用的 VID
Device Identification	DID	0x2A40	Contiga 使用的 DID
PCI Command	PCICMD	0x0006	支持存储器空间，可作为主设备
PCI Status	PCISTS	0x0090	支持背靠背传送和 Capability 寄存器
Revision Identification	RID	0x00	版本号为 0
Class Code	CC	0x060000	表示该设备为 Host Bridge
Master Latency Timer	MLT	0x00	PCIe 设备不再使用该寄存器
Header Type	HDR	0x00	表示为 PCI 单功能设备
Subsystem Vendor Identification	SVID	0x0000	未使用
Subsystem Identification	SID	0x0000	未使用
Capability Pointer	CAPPTR	0xE0	第一个 Capability 寄存器地址为 0xE0

Device 0 使用的基本配置空间与其他 PCI 设备兼容。这里值得注意的是 Device 0 在 PCIe 体系结构中，被认为是 HOST 主桥。而 Device 0 使用的 PCI 扩展配置空间也被称为 RCRB，RCRB 的主要作用是描述当前处理器的存储器地址拓扑结构，包括主存储器地址和 PCI 总线地址。其简写和复位值如表 5-2 所示。

表 5-2　Device 0 的扩展 PCI 配置空间

寄存器名	简　写	缺省值
Egress Port Base Address	EPBAR	0x0000-0000-0000-0000
（G）MCH Memory Mapped Register Range Base	MCHBAR	0x0000-0000-0000-0000

（续）

寄存器名	简　写	缺省值
（G）MCH Graphics Control Register	GGC	0x0030
Device Enable	DEVEN	0x000043DB
PCI Express Register Range Base Address	PCIEXBAR	0x0000-0000-E000-0000
MCH-ICH Serial Interconnect Ingress Root Complex	DMIBAR	0x0000-0000-0000-0000
Programmable Attribute Map 0	PAM0	0x00
Programmable Attribute Map 1	PAM1	0x00
Programmable Attribute Map 2	PAM2	0x00
Programmable Attribute Map 3	PAM3	0x00
Programmable Attribute Map 4	PAM4	0x00
Programmable Attribute Map 5	PAM5	0x00
Programmable Attribute Map 6	PAM6	0x00
Legacy Access Control	LAC	0x00
Remap Base Address Register	REMAPBASE	0x03FF
Remap Limit Address Register	REMAPLIMIT	0x0000
System Management RAM Control	SMRAM	0x02
Extended System Management RAM Control	ESMRAMC	0x38
Top of Memory	TOM	0x0001
Top of Upper Usable DRAM	TOUUD	0x0000
Top of Low Used DRAM Register	TOLUD	0x0010
Error Status	ERRSTS	0x0000
Error Command	ERRCMD	0x0000
Scratchpad Data	SKPD	0x0000-0000
Capability Identifier	CAPID0	0x0000-0000-0000-010A-0009

系统软件首先检查 Capability Identifier 寄存器，该寄存器的地址偏移为 0xE0，即 CAPPTR 寄存器指向的地址为 0xE0。该 Capability 结构使用的 PCI Express Extended Capability ID 字段（该字段在 CAPID0 寄存器中）为 0x0A，因此表 5-2 中的寄存器组为 RCRB Capability 结构，有关 Capability 结构的组成结构见第 4.3 节。

在 x86 处理器系统中，RCRB 存放一些与处理器系统相关的寄存器。而在许多处理器中，如在 PowerPC 处理器中并不含有 RCRB。在 PowerPC 处理器中，与处理器系统相关的寄存器都存放在以 BASE_ADDR 为起始地址的 1MB 连续的物理地址空间中，PowerPC 处理器系统使用存储器映射寻址方式访问这些寄存器。

在 x86 处理器系统中，使用 PCI 总线管理所有外部设备，这些“与处理器系统相关的寄存器”被保存在 RCRB 中，处理器使用 PCI 总线配置周期访问这些寄存器。实际上在 RCRB 中包含的寄存器与 PCIe 体系结构并没有直接关系，这些寄存器应该属于存储器域的地址区域。x86 处理器的这种做法并非完全合理，在某种程度上容易使初学者混淆存储器域与 PCI 总线域的区别。RCRB 主要寄存器的含义如下所示。

5.1.1 EPBAR 寄存器

EPBAR 寄存器的大小为 8B，指向一个 4KB 大小的存储器区域。处理器使用存储器映像寻址访问这段存储器区域，并通过这段存储器区域访问 RCRB 的扩展配置空间，在表 5-2 中存放的仅是 RCRB 的部分扩展配置空间。

这段存储器区域描述 RC 的 Egress 端口属性，包括 RC 使用的 VC0 和 VC1 两个虚通路的具体信息。当 EPBAR 寄存器的“EPBAR Enable”位为 1 时，这段空间有效。这段寄存器区域被称为“Egress Port RCRB”空间，系统软件可以使用这段空间定义的寄存器，完成对 VC1 和 VC0 通路的设置，包括端口仲裁、VC 仲裁等一系列的内容。

5.1.2 MCHBAR 寄存器

MCHBAR 寄存器的大小为 8 B，指向一个 16 KB 大小的存储器区域。处理器使用存储器映像寻址访问这段存储器区域。这段存储器区域描述 GMCH 内部使用的一些寄存器。当 MCHBAR 寄存器的“MCHBAR Enable”位为 1 时，这段存储器区域有效。

在这段区域中含有多组寄存器。

- Device 0 Memory Mapped I/O 寄存器组。包括 MEREMAPBAR、GFXREMAPBAR、VC0REMAPBAR、VC1REMAPBAR 和 PAVPC 寄存器。
- DRAM Channel Control 寄存器。该寄存器用来设置 x86 处理器系统中的 DRAM 通路。在 Montevina 平台中含有两个 DRAM 通路。这两个 DRAM 通路可以独立使用，也可以设置成 Interleaved 模式。
- MCHBAR Clock Control 寄存器组。由 CLKCFG 和 SSKPD 寄存器组成，其中 CLKCFG 寄存器可以设置 DDR 使用的频率和 FSB 总线的频率；而 SSKPD 寄存器供 BIOS 或者 Graphic 驱动使用，用来保存一些中间结果。
- Device 0 MCHBAR ACPI Power Management Controls 寄存器组。该组寄存器与 x86 处理器系统使用的 ACPI 有关。
- Device 0 MCHBAR Thermal Management Controls 和 MCHBAR Render Thermal Throttling 寄存器组。该组寄存器与 Montevina 平台中的温度传感器相关。
- Device 0 MCHBAR DRAM Controls 寄存器组。这组寄存器对 Montevina 平台中的两个 DRAM 通路进行细粒度的控制，包括这两个 DRAM 通路中使用的 Timing、延时和各种模式选择。该寄存器组对于需要设置 DRAM 属性的 BIOS 工程师非常重要。

5.1.3 其他寄存器

在 Device 0 中还包含以下寄存器。

- GGC 寄存器。在 x86 处理器系统中，显卡可以借用一部分存储器域的地址空间，该寄存器描述这段被借用的存储器地址空间。
- DEVEN 寄存器。该寄存器用来使能/禁止在 MCH 中的虚拟 PCI 设备，如显卡控制器和其他虚拟设备。
- PCIEXBAR 寄存器。该寄存器的大小为 8 B，指向一个 256 MB 大小的存储器区域。PCIe 总线可以使用 ECAM 方式访问 PCI 设备的扩展配置空间，PCIEXBAR 寄存器存放

PCI 配置空间的基地址，有关 ECAM 机制的详细信息见第 5.3.2 节。

- DMIBAR 寄存器。该寄存器的大小为 8 B，指向一个 4 KB 大小的存储器区域，处理器使用存储器映像寻址访问这段存储器区域。该寄存器描述 RC 中的 DMI 接口。
- PAM0 ~ PAM6 寄存器描述 Shadow BIOS 的属性。
- REMAPBASE 寄存器和 REMAPLIMIT 寄存器支持 x86 处理器的 Reclaim 机制，第 5.2 节将详细介绍该机制。
- SMRAM 寄存器和 ESMRAMC 寄存器控制处理器对 SMRAM 的访问。SMRAM 与 x86 处理器的 SMM 机制相关，本书对此不做介绍。
- TOM、TOUUD 和 TOLUD 寄存器与处理器系统的主存储器管理相关，分别描述存储器的物理大小和存储器“低于 4 GB”和“高于 4 GB”地址空间的使用方法。第 5.2.3 节将详细介绍这几个寄存器。

5.2 Montevina 平台的存储器空间的组成结构

由上文所述，在 Montevina 平台中包含一个 Mobile 处理器、MCH、ICH 和一个无线网卡适配器组成。在 MCH 和 ICH 中具有许多组成部件，本节仅介绍 MCH 和 ICH 中与 PCI 总线直接相关的部分内容。

Montevina 平台使用的地址空间由存储器域地址空间和 PCI 总线域地址空间组成。在 Intel 的 x86 处理器系统中，包括 Montevina 平台，所有的外部设备都通过 PCI 总线进行管理。x86 处理器平台使用这种方法便于对外部设备统一管理，但是这种方法也带来了一些弊端。因为使用这种方法时，PCI 总线域空间与存储器域空间的边界划分并不明晰。

Montivina 平台除了具有存储器域、PCI 总线域之外，还存在一个 DRAM 域。所谓 DRAM 域是指 DRAM 控制器所能访问的地址空间，即从 DRAM 控制器的角度来看，DRAM 空间的拓扑结构。DRAM 域中包含的地址空间，通俗地讲是指主存储器地址空间，即 DRAM 控制器能够访问的地址空间。

在 Montevina 平台中，DRAM 域地址空间并不能与存储器域地址空间完全对应。当处理器系统支持的内存超过 4 GB 时，DRAM 域的部分空间需要使用 Reclaim 机制才能访问，此外在 DRAM 域空间中，有些地址并不能被处理器访问。比如显卡控制器借用了一部分 DRAM 空间，这部分空间可以被显卡控制器访问，但是不能被 CPU 访问。x86 处理器由于考虑向前兼容，一个原本完整的 DRAM 域被划分得支离破碎。在 Intel 的 x86 处理器中，许多“不合理”都是因为“向前兼容”导致的。

在 x86 处理器系统中，存储器域由 CPU 能够访问的地址空间组成，包括 DRAM 域地址空间的一部分，一些使用存储器映像寻址的寄存器[㊀]和 PCI 总线域地址空间在存储器域中的映像。

而 DRAM 域由 DRAM 控制器所能寻址的空间组成。如图 5-2 所示，在 Montevina 平台中，存储器域与 DRAM 域的所包含的部分空间，其地址相等，比如 Legacy Address Range、TSEG（Top of Memory Segment）和其他一些 DRAM 空间。这里的地址相等指在存储器域和

㊀ x86 处理器也将使用存储器映像寻址的寄存器称为 MMIO（Memory Mapped I/O）。

DRAM 域中的地址相同，但是这两个地址的含义并不相同。

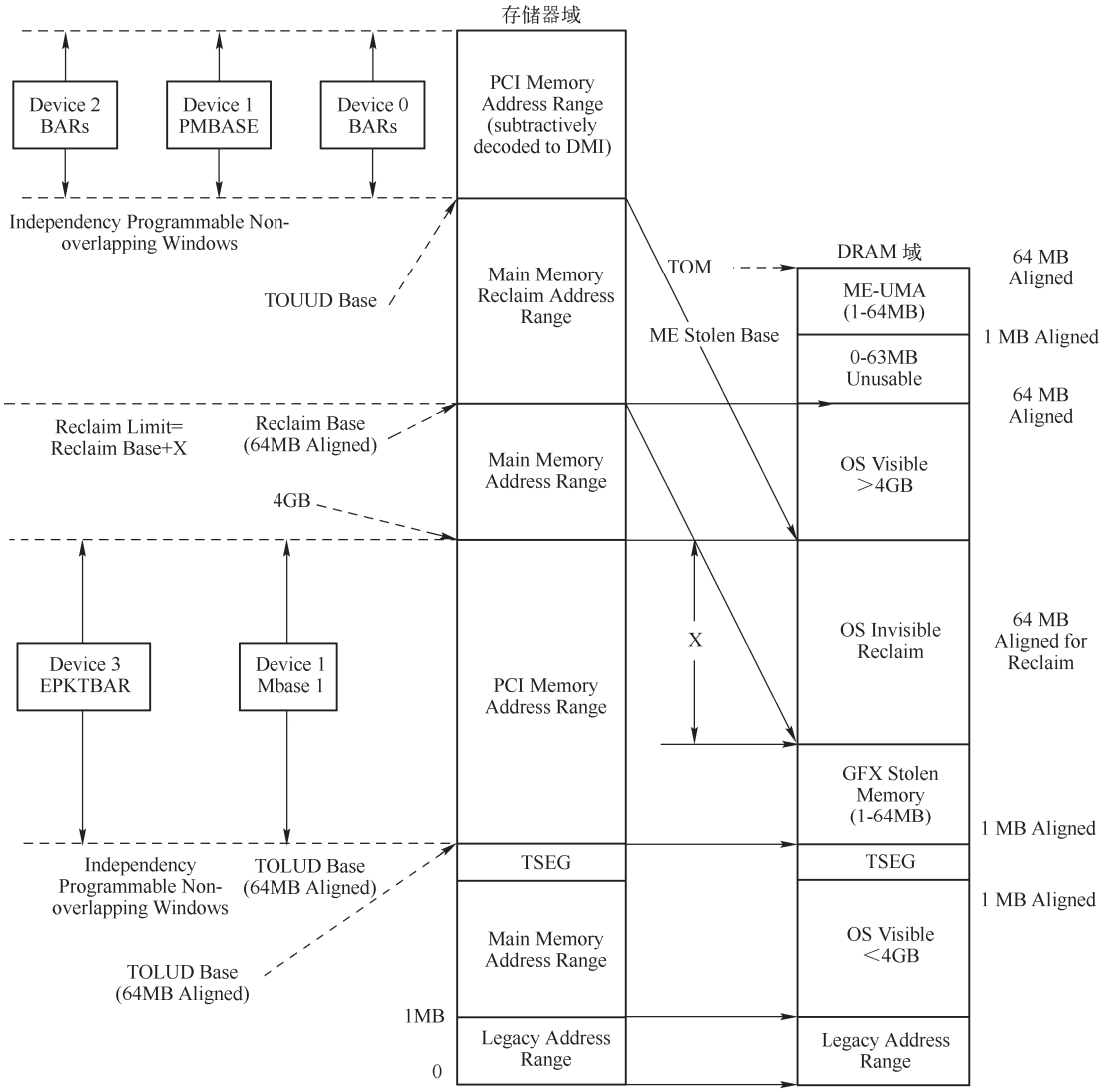

图 5-2　Montevina 平台的 CPU 域与 DRAM 域

在一个多核处理器系统中，不同的 CPU 所能访问的地址空间也不一定相同，其中每一个 CPU 都对应一个存储器域。在这些存储器域中，有些空间是所有 CPU 共享的，有些空间是某个 CPU 的私有空间。在多核处理器中，存储器域和 DRAM 域地址空间的划分更为复杂，本书对此不做进一步说明。

如图 5-2 所示，x86 处理器将 PCI 总线域和存储器域进行混合编址。但是在图 5-2 中的 PCI 总线地址仅是在存储器域中的地址，即 PCI 总线地址在存储器域地址空间的映像。值得注意的是，这个 PCI 总线地址和 PCI 总线域的地址没有直接联系，当处理器访问 PCI 设备时，首先使用在存储器域的 PCI 总线地址，RC 会将存储器域的地址转换为 PCI 总线域的地址，并使用 PCIe 总线事务访问相应的设备。

x86 处理器和 PowerPC 处理器进行存储器域到 PCI 总线域的映射方法不同。PowerPC 处理器使用 Inbound/Outbound 窗口显式地分离存储器域与 PCI 总线域。而 x86 处理器内部并没有设置这类寄存器显式分离这些域空间。但是 x86 处理器仍然区分存储器域和 PCI 总线域，虽然 PCI 设备使用的地址在存储器域和 PCI 总线域中相同。

x86 处理器采用的这种 PCI 总线域与其他处理器有较大的不同，x86 处理器采用这种设计方法可能考虑了向前兼容。采用这种结构，存储器域和 PCI 总线域之间的界限并不明晰，但是有利于外部设备的统一管理。

5.2.1 Legacy 地址空间

Legacy 地址空间在存储器域和 DRAM 域中都有映射，且地址相等，这段空间是 x86 处理器所固有的一段大小为 1 MB 的内存空间，它伴随着 x86 处理器的整个发展历程。Legacy 地址空间的大小为 1 MB，其详细说明如下。

- 0x0000-0000 ~ 0x0009-FFFF，DOS 使用的 640 KB 大小的基本内存空间。目前开发 BIOS 的工程师仍然在 DOS 下工作。
- 0x000A-0000 ~ 0x000B-FFFF，Legacy Vidio 空间，大小为 128 KB。这段空间作为 VGA 设备的 Frame Buffer。目前使用的高端显卡，处于 VGA 模式时，也使用这段空间作为 Frame Buffer。
- 0x000A-0000 ~ 0x000B-FFFF，Compatible SMRAM 空间，大小为 128 KB。
- 0x000B- 0000 ~ 0x000B-7FFF，MDA（Monochrome Adapter）空间，大小为 32 KB。MDA 是一种非常老的单色显示设备。
- 0x000C-0000 ~ 0x000D-FFFF，ISA 扩展区域，大小为 128 KB。
- 0x000E-0000 ~ 0x000E-FFFF，BIOS 扩展区域，大小为 128 KB。
- 0x000F-0000 ~ 0x000F-FFFF，系统 BIOS 区域，大小为 128 KB。

5.2.2 DRAM 域

在处理器系统中，可能含有多个 DRAM 控制器，可以管理多个 DRAM 空间，这些空间可以统一编址，也可以独立编址，从而组成一个或者多个 DRAM 域。DRAM 域的大小由一个处理器系统中具有的物理内存大小决定，在绝大多数处理器系统中，DRAM 域的物理地址空间是连续的，其最低地址为 0x0。DRAM 域地址空间的设置与处理器系统使用的 DRAM 控制器相关。在 PowerPC 处理器中，DRAM 域的最低物理地址是可变的，而且 DRAM 域的地址空间也可以不连续。

而在多数 x86 处理器中，DRAM 域的物理地址，其基地址不可改写，值为 0x0。多数 BIOS 都将 DRAM 域的地址空间设置为一段连续的地址空间，但是 x86 处理器也支持不连续的 DRAM 区域。x86 处理器系统的 DRAM 控制器远比 Power 处理器复杂，只是多数系统软件程序员并没有关心这些细节。

在 Montevina 平台中，DRAM 域空间的大小，即 x86 处理器使用的主存储器大小，保存在 TOM（Top of Memory）寄存器中，该寄存器在 MCH 的存储器控制器中，即 PCI 设备 0 的配置空间中。在 DRAM 域中除了 Legacy Address 空间之外，还包括以下几种空间。

1. GFX（显卡控制器）借用的空间

这段空间的大小为 1 MB ~ 64 MB，由 GFX 管理，CPU 不能访问这段空间。这段空间的基地址保存在 GBSM（Graphics Base of Stolen Memory）寄存器中，其值为 TOLUD（Top of Low Usable DRAM）寄存器的值。

TOLUD 和 GBSM 寄存器在存储器控制器中，其中 TOLUD 寄存器保存 x86 处理器系统低端内存（小于 4 GB）的大小。TOLUD 寄存器的值由 BIOS 根据处理器系统的配置决定，该寄存器的详细描述见下文。

2. TSEG 空间

TSEG 空间，这段空间的大小为 1 MB ~ 8 MB。这段空间仅在 CPU 进入 SMM 模式时，才能够被 CPU 访问。这段空间的基地址保存在 TSEGMB（TESG Memory Base）寄存器中，其值为 TOLUD 寄存器减去 GFX 借用空间和 TESG 空间的大小。TSEGMB 寄存器也保存在存储器控制器中。在 x86 处理器中，还有一段进入 SMM 模式才能访问的地址空间，HSEG 空间。这段空间被处理器映射到高端地址，HSEG 和 TSEG 空间的区别在于 TSEG 空间是可 Cache 的，而 HSEG 空间不可 Cache。

3. 小于 4 GB 的操作系统可用内存空间

在 Montevina 平台中，如果主存储器的大小超过 4 GB，那么这段物理空间将被分为三段供操作系统使用，分别是小于 4 GB 的空间、大于 4 GB 的空间和 Reclaim 空间。

如图 5-2 所示，这几段空间在存储器域中的地址并不连续。但是在 DRAM 域中，这些地址空间是连续，并从地址 0 到 TOM。而在存储器域中，小于 4 GB 的内存空间从 1 MB 开始到 TOLUD 结束。

4. OS Invisible Reclaim 空间

对于 DRAM 控制器而言，这段空间是存在的，不过因为这段地址和存储器域中的 PCI 总线地址空间重合，因此 CPU 不能直接访问这段物理内存。当处理器使能 Reclaim 机制（也称为 Remapping 机制）后，CPU 才能访问这段地址空间。

在一个 x86 处理器系统中，当实际的物理内存大于 TOLUD 加上 GFX 借用空间的大小时，DRAM 域中才含有这段空间，此时 CPU 才有必要启动 Reclaim 机制，将这段内存地址空间映射到存储器域中。当然 CPU 也可以不启动 Reclaim 机制，放弃对这段内存空间的使用。

5. 大于 4 GB 的操作系统可用空间

这段空间在存储器域和 DRAM 域中的地址相等，只有实际的物理内存大于 4 GB 时，存储器域和 DRAM 域中才有这段空间。

6. ME（Manageability Engine）-UMA 借用的空间

其大小为 1 ~ 64 MB。其基地址为 TOM 寄存器中的值减去 ME-UMA 借用空间的大小。由于 TOM 寄存器中的值需要 64 MB 对界，因此有时在其下会出现一个 0 ~ 63 MB 大小的空洞。这段空洞是被浪费的空间，除了 DRAM 控制器可以访问这段空间外，在处理器系统中的其他部件均不能访问这段空间。

5.2.3 存储器域

在 x86 处理器系统中，除了具有 DRAM 域地址空间外，还具有 PCI 地址空间。CPU 访问这些地址空间时，需要进行地址转换，CPU 访问 DRAM 域时，需要进行存储器域地址空

间到 DRAM 域地址空间的转换；CPU 访问 PCI 总线域时，需要进行存储器域地址空间到 PCI 总线域地址空间的转换。在 x86 处理器中，大多数 DRAM 域中的地址与存储器域中的地址一一对应而且相等，而存储器域的 PCI 地址与 PCI 总线域的地址也一一对应而且相等。

在 x86 处理器中，并没有提及地址转换部件，但是这个地址转换部件的概念是存在于芯片设计中的。CPU 访问 DRAM 或者 PCI 设备时，首先需要访问存储器域的地址，这些发送到存储器域的数据请求首先通过地址转换部件，并由地址转换部件决定这些数据请求是发向 PCI 总线域、DRAM 域还是其他域空间。在 x86 处理器中，这些地址域的关系如图 5-3 所示。

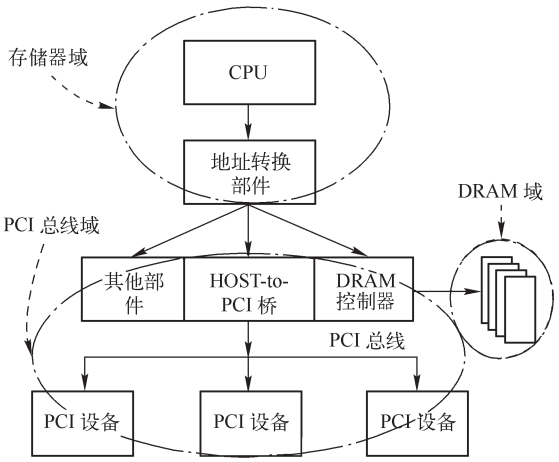

图 5-3　存储器域与 PCI/DRAM 域

在 Montevina 平台中，存储器域中的 Legacy Address 空间的地址与 DRAM 域空间的地址一一对应，而且地址相等。当 CPU 访问这段空间时，地址转换部件直接将发向存储器域的数据请求转发到 DRAM 域。而 CPU 访问其他段空间时，需要分别进行处理。

1. 1 MB ~ TOLUD 空间

这段空间由三部分组成，GFX Stolen Memory 空间、TSEG 和 Main Memory Address 空间。当 CPU 访问 GFX Stolen Memory 空间时，地址转换部件将拒绝这次访问，因为 GFX Stolen Memory 空间是 GFX 控制器的私有空间，CPU 不能对这段空间进行操作。

TSEG 空间只能在 CPU 处于 SMM 状态时才能访问，否则处理器内部的地址转换部件也将拒绝这次访问。

Main Memory Address 空间与 DRAM 域中小于 4 GB 的操作系统可用空间一一对应且地址相等，当 CPU 访问这段地址空间时，地址转换部件直接将这次访问转发到 DRAM 域，并由存储器控制器访问这段空间。这段空间也是存储器域中第 1 段“主存储器”空间。

2. TOLUD ~ 4 GB 空间

这段空间是存储器域中的 PCI 总线地址空间，当 CPU 访问这段地址空间时，地址转换部件将这次访问转发给 HOST 主桥或者 RC，之后由 HOST 主桥或者 RC 将 CPU 的这次数据访问转化为 PCI 总线的总线事务，之后再发送到对应的 PCI 设备。在 x86 处理器中，存储器域的 PCI 总线地址空间与 PCI 总线域的地址空间一一对应，其值完全相等。一些采用存储器映像寻址的寄存器也存放在这段空间中，如 APIC 中断控制器使用的地址空间。

3. 4 GB ~ TOUUD 空间

这段空间由 Main Memory Address 空间和 Main Memory Reclaim Address 空间组成。如图 5-2 所示，在一个 x86 处理器系统中，如果实际的物理内存空间大于 4 GB 时，存储器域中将含有第 2 段 Main Memory Address 空间。

这段空间与 DRAM 域中大于 4 GB 的操作系统可用空间一一对应，而且地址相等，CPU 访问这段地址空间时，地址转换部件直接将这次访问转发到 DRAM 域，这段空间的上界为

TOM 减去 ME（Manageability Engine）借用的空间。ME 借用的地址空间不能被 CPU 访问，本节对这段空间不做进一步介绍。

4. Reclaim 空间

如果 CPU 使能了 Reclaim 机制，则存储器域具有这段空间。在 x86 处理器中存储器域与 DRAM 域基本上一一对应而且地址相等，但是 TOLUD ~ 4 GB 这段空间例外。存储器域将这段空间分配为 PCI 总线地址空间，因此 CPU 不能直接用“相同的地址”访问 DRAM 这段空间。只有 x86 处理器使用 Reclaim 机制，将 DRAM 域的“OS Invisible Reclaim 空间”映射到存储器域的“Reclaim Base ~ TOUUD”这段空间后，处理器才能访问这段空间。

这段空间也是 x86 处理器的第 3 段存储器空间。值得注意的是这段存储器空间在存储器域中的地址与 DRAM 域中的地址并不相等。

当 Reclaim 机制使能后，Reclaim Base 的值为 TOM 减去 ME 借用的空间，这段空间的大小等于 4 GB 减去 TOLUD（64 MB 对界），然后再减去 GFX 借用的空间大小。此时 TOUUD 的值等于 Reclaim Base 加上 Reclaim 空间的大小。

5. TOUUD ~ 64 GB 空间

在 x86 处理器系统中，PCI 总线需要的空间大于 TOLUD ~ 4 GB 这段区域时，将使用这段空间映射剩余的 PCI 总线地址空间，这段空间的使用方法与 BIOS 的设置有关。在 Montevina 平台中，CPU 使用的物理地址为 36 位，因此存储器域的最大物理地址空间为 64 GB。

5.3 存储器域的 PCI 总线地址空间

由图 5-2 可知，Montevina 平台中存储器域的 PCI 总线地址空间分为 TOLUD ~ 4 GB 和 TOUUD ~ 64 GB 这两段空间。这两段空间都可以映射 PCI 设备使用的空间，x86 处理器为了实现向前兼容，并没有取消 TOLUD ~ 4 GB 这段空间，而这段空间的存在将存储器域的 DRAM 空间一分为二。

5.3.1 PCI 设备使用的地址空间

TOLUD ~ 4 GB 这段 PCI 总线地址空间主要映射和 ICH 相连的 PCI 设备地址空间，此外还包括 EPBAR（Egress Port Base Address）指向的空间，以及 MCHBAR 和 DMIBAR 指向的空间等。除了 PCI 总线地址空间外，这段空间还包括 High BIOS、APIC（Advanced Programming Interrupt Controller）和 FSB Interrupts 地址空间，其详细描述如图 5-4 所示。

其中 APIC（Advanced Programmable Interrupt Controller）包括 I/O APIC 和 Local APIC 占用 0xFEC0-0000 ~ 0xFECF-FFFF 这段地址空间。APIC 是 x86 处理器使用的中断控制器，负责管理外部和 CPU 之间的中断请求。而 HESG 空间占用 0xFEDA-0000 ~ 0xFEDB-FFFF 这段地址空间，这段空间在 CPU 内核处于 SMM 状态时，才能访问。

FSB Interrupts 存储器空间与 MSI 中断机制相关，PCIe 设备向这段存储器空间进行写操作时，MCH 将这个写操作转换为 FSB 总线的 Interrupt Message 总线事务。

值得注意的是，在 x86 处理器中 Local APCI 使用的寄存器在 0xFEE0-0000 ~ 0xFEE0-03F0 区域之间，在这段区域中，有一些 Reserved 寄存器，如 0xFEE0-0000 ~ 0xFEE0-0010，系统软件不能操作这些寄存器。Intel 并没有公开这些寄存器的具体含义，但是从原理上推

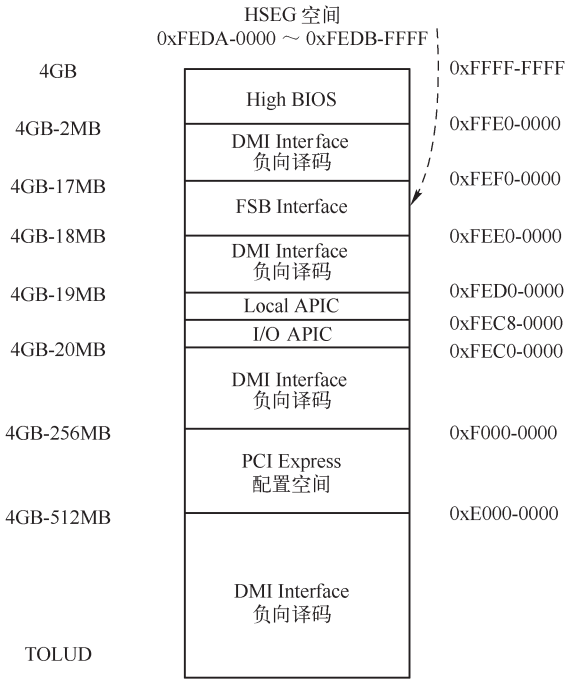

图 5-4　TOLUD ~ 4 GB PCI 总线地址空间

断，这段寄存器所使用的地址正好是 PCIe 设备使用 MSI-X 中断方式向 APCI ID 为 0 的 CPU 发送中断所使用的地址，有关 x86 处理器 MSI-X 中断方式的详细说明见第 10. 3 节。

在 x86 处理器中，Local APIC 寄存器空间是可变的，其基地址保存在 IA32_APIC_BASE 寄存器中，该寄存器是 x86 处理器的 MSR（Model Specific Register）寄存器，x86 处理器使用 RDMSR 和 WRMSR 指令访问这些寄存器。

PCI Express 配置空间占用 0xE000-0000 ~ 0xEFFF-FFFF 这段地址空间，下节将详细解释这段空间的使用方法。DMI Interface 负向译码空间被分为若干段，用来映射 ICH 使用的 PCI 总线地址空间。在 ICH 中提供了许多 PCI 设备，包括内嵌在 ICH 中的虚拟 PCI 设备和 PCIe 总线端口。这些 PCI 设备的 BAR 空间被映射到这段空间。

Montevina 平台使用 DMI 连接 MCH 和 ICH。当 CPU 对 PCI 空间发起数据请求时，这些数据首先到达 MCH，当 MCH 中的所有设备都不响应这个数据请求时，MCH 中的 DMI 接口设备将使用负向译码方式被动地接收这个数据请求，并将其转发到 ICH 中的 DMI 接口设备，从而到达 ICH。因此 Montevina 平台将这段空间称为“负向译码空间”。由以上说明可以发现与 MCH 连接的 PCIe 设备的访问延时小于与 ICH 连接中的 PCIe 设备。

5. 3. 2　PCIe 总线的配置空间

x86 处理器使用了两种机制访问 PCIe 设备的扩展配置空间。首先 x86 处理器提供了两个 I/O 端口寄存器，CONFIG_ADDRESS 和 CONFIG_DATA 寄存器，使用这两个寄存器访问 EP 配置空间的方法与访问 PCI 设备类似，详见第 2. 2. 4 节。然而这两个寄存器只能访问 PCI 设备的基本配置空间，即 PCIe 设备配置空间的前 256 个字节，而之后的扩展配置空间需要通

过 ECAM 方式进行访问。

在 Montevina 平台中，PCIe 设备配置空间的基地址保存在 PCIEXBAR 寄存器中，这段地址空间的大小为 256 MB，且为 256 MB 对界。当 CPU 对 PCIe 设备配置空间㊀进行读写访问时，MCH 将这个存储器读写请求转换为 PCI 配置读写总线周期后，再发送到相应的 PCIe 设备中。PCIe 总线规范将这种“对 PCIe 设备配置空间”的读写访问方式称为 ECAM 机制。

ECAM 机制的主要原理是，将处理器系统中所有 PCIe 设备的配置空间映射到一段地址连续的存储器域的地址空间中。CPU 可以直接对这段特殊的存储器域地址空间进行访问，从而访问 PCIe 设备的配置空间。

使用 ECAM 机制与使用 CONFIG_ADDRESS 和 CONFIG_DATA 这对寄存器，间接访问 PCIe 设备的配置空间有较大的不同。ECAM 机制是一种直接寻址方式，在 x86 处理器系统中，只有使用 ECAM 方式才可以访问 PCIe 设备的扩展配置空间。而其他处理器，如 PowerPC 处理器，即便不使用 ECAM 方式，也可以访问 PCIe 设备的扩展配置空间，在 MPC8548 处理器的 PEX_CONFIG_ADDR 寄存器㊁中，设置了 EXT_REGN 字段（由 4 位组成），该字段可以与 REGN 字段组成一个 10 位的字段，从而可以访问所有扩展的 PCIe 设备配置空间。

Linux 系统定义了 raw_pci_read 和 raw_pci_write 两个函数，对 PCI 设备配置空间进行读写，这两个函数的实现在 ./arch/x86/pci/common.c 文件中，如源代码 5-1 所示。

源代码 5-1　raw_pci_read 和 raw_pci_write 函数

```
int raw_pci_read(unsigned int domain, unsigned int bus, unsigned int devfn,
                        int reg, int len, u32 *val)
{
    if (domain == 0 && reg < 256 && raw_pci_ops)
        return raw_pci_ops->read(domain, bus, devfn, reg, len, val);
    if (raw_pci_ext_ops)
        return raw_pci_ext_ops->read(domain, bus, devfn, reg, len, val);
    return -EINVAL;
}

int raw_pci_write(unsigned int domain, unsigned int bus, unsigned int devfn,
                        int reg, int len, u32 val)
{
    if (domain == 0 && reg < 256 && raw_pci_ops)
        return raw_pci_ops->write(domain, bus, devfn, reg, len, val);
    if (raw_pci_ext_ops)
        return raw_pci_ext_ops->write(domain, bus, devfn, reg, len, val);
    return -EINVAL;
}
```

㊀ 这段空间属于存储器域地址空间。

㊁ 该寄存器与 PCI_CONFIG_ADDR 寄存器类似，用来访问 PCIe 设备的配置空间。

由以上代码，可以发现当 raw_pci_read/write 函数访问的配置寄存器号大于 256 B 时，该函数将调用 raw_pci_ext_ops 函数，否则调用 raw_pci_ops 函数。其中 raw_pci_ops 函数指针使用 0xCF8 和 0xCFC 两个 I/O 端口寄存器访问 PCI 总线配置空间，而 raw_pci_ext_ops 函数使用 ECAM 方式访问 PCI 总线配置空间。

在 Linux x86 系统中，raw_pci_ext_ops 函数相当于 pci_mmcfg 函数，而 pci_mmcfg 函数指针分别指向两个函数，为 pci_mmcfg_read 和 pci_mmcfg_write 函数。这两个函数使用直接寻址方式（即 ECAM 方式）访问当前处理器系统中所有 PCIe 设备的扩展配置空间，其函数原型在 ./arch/x86/pci/mmconfig_32. c 文件中。

Montevina 平台使用 256 MB 物理空间映射 PCI 设备的配置寄存器，因为在 Montevina 平台中有一棵 PCI 总线树，因此最多具有 256 条 PCI 总线，每一条 PCI 总线最多可以挂接 32 个 PCI 设备，每一个设备最多有 8 个 Function，而在每一个 Function 中最大的 PCI 总线配置寄存器空间为 4 KB。

因此将一棵 PCI 总线树上所有 PCI 设备的配置空间采用一一映射的方式对应到存储器域空间时，共需要 1 MB 空间，而一个 HOST 主桥可以管理的 PCI 总线为 256 条。为此 Montevina 平台共提供了 256 MB 大小的空间，其基地址在 PCIEXBAR 寄存器中，这段存储器域的地址空间与 PCI 总线的配置寄存器的对应关系如表 5-3 所示。

表 5-3　PCIEXBAR 空间与 PCI 总线配置寄存器的对应关系

地 址 偏 移	PCI 总线的配置空间
A［32:28］	与 PCIEXBAR 寄存器一致
A［27:20］	Bus Number
A［19:15］	Device Number
A［14:12］	Function Number
A［11:8］	Extended Register Number
A［7:2］	Register Number
A［1:0］	用作字节使能

当 CPU 对 PCIEXBAR 地址空间进行访问时，MCH 或者 ICH 将根据上表所示的规则，将这次存储器访问转化对 PCI 总线的某个设备进行的配置读写访问。如果当 CPU 对 PCIEXBAR +0x0811 -0000 这个地址进行访问时，MCH 将这次地址访问转换为对 Bus 号为 0x81（A［27:20］为 0x81），Device 号为 1（A［19:15］为 1），且 Function 号为 0 的 PCI 设备配置空间的访问，访问的寄存器为 0x0（A［11:2］为 0）。

虽然采用 ECAM 方式可以访问之前使用 CONFIG_DATA 和 CONFIG_ADDRESS 寄存器不能访问的 PCI 扩展配置寄存器空间，但是也带来了一些问题。在一个处理器系统中，同一条 PCI 总线上的 Device Number 不一定连续，而且在多数 PCI 设备中也不会有 8 个 Function，这将造成 PCIEXBAR 空间会留有许多空洞。虽然 Montevina 平台提供了 256 MB 的 PCIe 设备使用的配置空间，但是这些空间的实际利用率较低。

PowerPC 处理器也可以使用 ECAM 方式映射 PCI 配置空间，如 MPC8548 处理器可以使用 PCIe 桥中的 Outbound 寄存器（PEXOWARn）将 PCI 配置空间映射到存储器域。此外 PowerPC 处理器还可以使用 PEX_CFG_DATA 和 PEX_CFG_ADDR 这两个寄存器访问扩展

PCIe 设备的配置空间。其中 PEX_CONFIG_ADDR 寄存器的结构如图 5-5 所示。

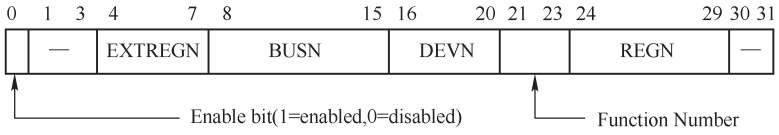

图 5-5 PEX_CONFIG_ADDR 寄存器的结构

PEX_CONFIG_ADDR 寄存器保存当前 PCIe 设备在处理器系统中的 ID 号，该寄存器的各个字段的描述如下。

- Enable 位。该位为 1 表示使能 HOST 处理器对 PCI 配置空间的访问，当 HOST 处理器对 PEX_CONFIG_DATA 寄存器进行访问时，HOST 主桥将对这个寄存器的访问转换为 PCI 配置读写周期并发送到 PCI 总线上。
- BUSN 字段记录 PCI 设备所在的总线号。
- DEVN 字段记录 PCI 设备的设备号。
- FUNCN 字段记录 PCI 设备的功能号。
- EXTREGN 和 REGN 字段，共 10 位，记录 PCI 设备的配置寄存器号。PowerPC 处理器使用这两个字段组成 10 位地址空间，从而可以访问 PCIe 设备使用的全部配置空间。

从原理上讲，x86 处理器也可以使用这种方式访问扩展的 PCI 配置空间，但是 x86 处理器没有采用这种方式，而是使用 ECAM 机制访问扩展的配置空间。这种方式将不可避免地在存储器域占用一段专用的地址空间。而这段地址空间的使用效率较低，因为在一条 PCI 总线上，PCIe 设备不会使用所有的设备号，而且每一个 PCIe 设备也不会使用所有的 Function 号，因此在这段地址空间中，有许多空间没有被充分利用。

5.4 小结

本章较为详细地介绍了 Montevina 平台与 PCIe 总线相关的内容。理解 PCIe 总线必须建立在理解处理器系统的基础之上，而 Intel 提供的处理器平台无疑是学习 PCIe 总线最合适的平台。Intel 是 PCIe 总线的缔造者，而且总是率先支持 PCIe 规范提出的最新功能。希望读者能够在充分理解处理器平台的基础上，进一步理解 PCIe 总线的细节知识。

本章因为篇幅有限，并不能对 Intel 的处理器平台进行详细分析与介绍。对这部分内容有兴趣的读者，可以首先阅读 Intel 提供的 Intel 64 and IA32 Architectures Software Developer's Manual 丛书（可以在 http://www.intel.com/products/processor/manuals 中下载）。

读者在获得这些入门知识之后，可以进一步阅读与 Intel 处理器相关的其他知识。目前 Intel 并没有完全公开其处理器平台的详细资料。但是从已有的资料中，也可以看到 Intel 在处理器领域的成就。目前 Intel 在这个领域处于无可争议的领袖地位。Intel 的处理器平台因为向前兼容的缘故，有些部件的实现并不完美，这些不完美是历史原因造成的。

第 6 章　PCIe 总线的事务层

事务层是 PCIe 总线层次结构的最高层，该层次将接收 PCIe 设备核心层的数据请求，并将其转换为 PCIe 总线事务，PCIe 总线使用的这些总线事务在 TLP 头中定义。PCIe 总线继承了 PCI/PCI-X 总线的大多数总线事务，如存储器读写、I/O 读写、配置读写总线事务，并增加了 Message 总线事务和原子操作等总线事务。

本节重点介绍与数据传送密切相关的总线事务，如存储器、I/O、配置读写总线事务。在 PCIe 总线中，Non-Posted 总线事务分两部分进行，首先是发送端向接收端提交总线读写请求，之后接收端再向发送端发送完成（Completion）报文。PCIe 总线使用 Split 传送方式处理所有 Non-Posted 总线事务，存储器读、I/O 读写和配置读写这些 Non-Posted 总线事务都使用 Split 传送方式。PCIe 的事务层还支持流量控制和虚通路管理等一系列特性，而 PCI 总线并不支持这些新的特性。

在 PCIe 总线中，不同的总线事务采用的路由方式不相同。PCIe 总线继承了 PCI 总线的地址路由和 ID 路由方式，并添加了“隐式路由”方式。

PCIe 总线使用的数据报文首先在事务层中形成，这个数据报文也称为事务层数据报文，即 TLP。TLP 在经过数据链路层时被加上 Sequence Number 前缀和 CRC 后缀，然后发向物理层。

数据链路层还可以产生 DLLP（Data Link Layer Packet）。DLLP 和 TLP 没有直接关系。DLLP 是产生于数据链路层，终止于数据链路层，并不会传递到事务层。DLLP 不是 TLP 加上前缀和后缀形成的。数据链路层的报文 DLLP 通过物理层时，需要经过 8/10b 编码，然后再进行发送。而数据的接收过程是发送过程的逆过程。

6.1　TLP 的格式

当处理器或者其他 PCIe 设备访问 PCIe 设备时，所传送的数据报文首先通过事务层被封装为一个或者多个 TLP，之后才能通过 PCIe 总线的各个层次发送出去。TLP 的基本格式如图 6-1 所示。

图 6-1　TLP 的格式

一个完整的 TLP 由 1 个或者多个 TLP Prefix、TLP 头、Data Payload（数据有效负载）和 TLP Digest 组成。TLP 头是 TLP 最重要的标志，不同的 TLP 其头的定义并不相同。TLP 头包含了当前 TLP 的总线事务类型、路由信息等一系列信息。在一个 TLP 中，Data Payload 的长度可变，最小为 0，最大为 1024DW。

TLP Digest 是一个可选项，一个 TLP 是否需要 TLP Digest 由 TLP 头决定。Data Payload 也是一个可选项，有些 TLP 并不需要 Data Payload，如存储器读请求、配置和 I/O 写完成 TLP 并不需要 Data Payload。

TLP Prefix 由 PCIe V2.1 总线规范引入，分为 Local TLP Prefix 和 EP-EP TLP Prefix 两类。其中 Local TLP Prefix 的主要作用是在 PCIe 链路的两端传递消息，而 EP-EP TLP Prefix 的主要作用是在发送设备和接收设备之间传递消息。设置 TLP Prefix 的主要目的是为了扩展 TLP 头，并以此支持 PCIe V2.1 规范的一些新的功能。

TLP 头由 3 个或者 4 个双字（DW）组成。其中第一个双字中保存通用 TLP 头，其他字段与通用 TLP 头的 Type 字段相关。一个通用 TLP 头由 Fmt、Type、TC、Length 等字段组成，如图 6-2 所示。

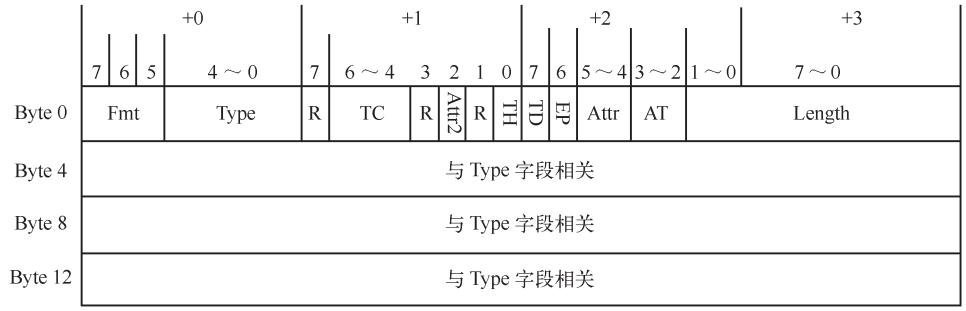

图 6-2　通用 TLP 头格式

如果存储器读写 TLP 支持 64 位地址模式时，TLP 头的长度为 4DW，否则为 3DW。而完成报文的 TLP 头不含有地址信息，使用的 TLP 头长度为 3DW。其中 Byte 4 ~ Byte 15 的格式与 TLP 相关，下文将结合具体的 TLP 介绍这些字段。

6.1.1　通用 TLP 头的 Fmt 字段和 Type 字段

Fmt 和 Type 字段确认当前 TLP 使用的总线事务，TLP 头的大小是由 3 个双字还是 4 个双字组成，当前 TLP 是否包含有效负载。其具体含义如表 6-1 所示。

表 6-1　Fmt［1:0］字段

Fmt［2:0］	TLP 的格式
0b000	TLP 大小为 3 个双字，不带数据
0b001	TLP 大小为 4 个双字，不带数据
0b010	TLP 大小为 3 个双字，带数据
0b011	TLP 大小为 4 个双字，带数据
0b100	TLP Prefix
其他	PCIe 总线保留

其中所有读请求 TLP 都不带数据，而写请求 TLP 带数据，而其他 TLP 可能带数据也可能不带数据，如完成报文可能含有数据，也可能仅含有完成标志而并不携带数据。在 TLP 的 Type 字段中存放 TLP 的类型，即 PCIe 总线支持的总线事务。该字段共由 5 位组成，其含

义如表6-2所示。

表6-2 Type［4:0］字段

TLP类型	Fmt［2:0］	Type［4:0］	描　　述
MRd	0b000 0b001	0b0 0000	存储器读请求；TLP头大小为3个或者4个双字，不带数据
MRdLk	0b000 0b001	0b0 0001	带锁的存储器读请求；TLP头大小为3个或者4个双字，不带数据
MWr	0b010 0b011	0b0 0000	存储器写请求；TLP头大小为3个或者4个双字，带数据
IORd	0b000	0b0 0010	IO读请求；TLP头大小为3个双字，不带数据
IOWr	0b010	0b0 0010	IO写请求；TLP头大小为3个双字，带数据
CfgRd0	0b000	0b0 0100	配置0读请求；TLP头大小为3个双字，不带数据
CfgWr0	0b010	0b0 0100	配置0写请求；TLP头大小为3个双字，带数据
CfgRd1	0b000	0b0 0101	配置1读请求；不带数据
CfgWr1	0b010	0b0 0101	配置1写请求；带数据
TCfgRd	0b010	0b1 1011	本书对这两种总线事务不做介绍
TCfgWr	0b001	0b1 1011	
Msg	0b001	0b1 $0r_2r_1r_0$	消息请求；TLP头大小为4个双字，不带数据。“rrr”字段是消息请求报文的Route字段，下文将详细介绍该字段
MsgD	0b011	0b1 $0r_2r_1r_0$	消息请求；TLP头大小为4个双字，带数据
Cpl	0b000	0b0 1010	完成报文；TLP头大小为3个双字，不带数据。包括存储器、配置和I/O写完成
CplD	0b001	0b0 1010	带数据的完成报文，TLP头大小为3个双字，包括存储器读、I/O读、配置读和原子操作读完成
CplLk	0b000	0b0 1011	锁定的完成报文，TLP头大小为3个双字，不带数据
CplDLk	0b010	0b0 1011	带数据的锁定完成报文，TLP头大小为3个双字，带数据
FetchAdd	0b010 0b011	0b0 1100	Fetch and Add原子操作
Swap	0b010 0b011	0b0 1101	Swap原子操作
CAS	0b010 0b011	0b0 1110	CAS原子操作
LPrfx	0b100	0b0 $L_3L_2L_1L_0$	Local TLP Prefix
EPrfx	0b100	0b1 $E_3E_2E_1E_0$	End-End TLP Prefix

存储器读和写请求，IO读和写请求，及配置读和写请求的type字段相同，如存储器读和写请求的Type字段都为0b0 0000。此时PCIe总线规范使用Fmt字段区分读写请求，当Fmt字段是“带数据”的报文，一定是“写报文”；当Fmt字段是“不带数据”的报文，一定是“读报文”。

PCIe总线的数据报文传送方式与PCI总线数据传送有类似之处。其中存储器写TLP使用Posted方式进行传送，而其他总线事务使用Non-Posted方式。

PCIe 总线规定所有 Non-Posted 存储器请求使用 Split 总线方式进行数据传递。当 PCIe 设备进行存储器读、I/O 读写或者配置读写请求时，首先向目标设备发送数据读写请求 TLP，当目标设备收到这些读写请求 TLP 后，将数据和完成信息通过完成报文（Cpl 或者 CplD）发送给源设备。

其中存储器读、I/O 读和配置读需要使用 CplD 报文，因为目标设备需要将数据传递给源设备；而 I/O 写和配置写需要使用 Cpl 报文，因为目标设备不需要将任何数据传递给源设备，但是需要通知源设备，写操作已经完成，数据已经成功地传递给目标设备。

在 PCIe 总线中，进行存储器或者 I/O 写操作时，数据与数据包头一起传递；而进行存储器或者 I/O 读操作时，源设备首先向目标设备发送读请求 TLP，而目标设备在准备好数据后，向源设备发出完成报文。

PCIe 总线规范还定义了 MRdLk 报文，该报文的主要作用是与 PCI 总线的锁操作相兼容，但是 PCIe 总线规范并不建议用户使用这种功能，因为使用这种功能将极大影响 PCIe 总线的数据传送效率。

与 PCI 总线不同，PCIe 总线规范定义了 Msg 报文，即消息报文，分别为 Msg 和 MsgD。这两种报文的区别在于一个报文可以传递数据，一个不能传递数据。

PCIe V2.1 总线规范还补充了一些总线事务，如 FetchAdd、Swap、CAS、LPrfx 和 EPrfx。其中 LPrfx 和 EPrfx 总线事务分别与 Local TLP Prefix 和 EP-EP TLP Prefix 对应。在 PCIe 总线规范 V2.0 中，TLP 头的大小为 1DW，而使用 LPrfx 和 EPrfx 总线事务可以对 TLP 头进行扩展，本节不对这些 TLP Prefix 做进一步介绍。PCIe 设备可以使用 FetchAdd、Swap 和 CAS 总线事务进行原子操作，第 6.3.5 节将详细介绍该类总线事务。

6.1.2 TC 字段

TC 字段表示当前 TLP 的传送类型，PCIe 总线规定了 8 种传输类型，分别为 TC0 ~ TC7，缺省值为 TC0，该字段与 PCIe 的 QoS 相关。PCIe 设备使用 TC 区分不同类型的数据传递，而多数 EP 中只含有一个 VC，因此这些 EP 在发送 TLP 时，也仅仅使用 TC0，但是有些对实时性要求较高的 EP 中，含有可以设置 TC 字段的寄存器。

在 Intel 的高精度声卡控制器（High Definition Audio Controller）的扩展配置空间中含有一个 TCSEL 寄存器。系统软件可以设置该寄存器，使声卡控制器发出的 TLP 使用合适的 TC。声卡控制器可以使用 TC7 传送一些对实时性要求较强的控制信息，而使用 TC0 传送一般的数据信息。在具体实现中，一个 EP 也可以将控制 TC 字段的寄存器放入到设备的 BAR 空间中，而不必和 Intel 的高精度声卡控制器相同，存放在 PCI 配置空间中。

目前许多处理器系统的 RC 仅支持一个 VC 通路，此时 EP 使用不同的 TC 进行传递数据的意义不大。x86 处理器的 MCH 中一般支持两个 VC 通路，而多数 PowerPC 处理器仅支持一个 VC 通路。PLX 公司的多数 Switch 也仅支持两个 VC 通路。

有些 RC，如 MPC8572 处理器，也能决定其发出 TLP 使用的 TC。在该处理器的 PCIe Outbound 窗口寄存器（PEXOWARn）中，含有一个 TC 字段，通过设置该字段可以确定 RC 发出的 TLP 使用的 TC 字段。不同的 TC 可以使用 PCIe 链路中的不同 VC，而不同的 VC 的仲裁级别并不相同。EP 或者 RC 通过调整其发出 TLP 的 TC 字段，可以调整 TLP 使用的 VC，从而调整 TLP 的优先级。

6.1.3 Attr 字段

Attr 字段由 3 位组成，其中第 2 位表示该 TLP 是否支持 PCIe 总线的 ID-based Ordering；第 1 位表示是否支持 Relaxed Ordering；而第 0 位表示该 TLP 在经过 RC 到达存储器时，是否需要进行 Cache 共享一致性处理。Attr 字段如图 6-3 所示。

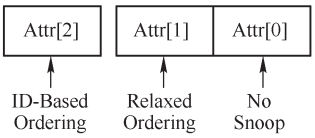

图 6-3　Attr 字段格式

一个 TLP 可以同时支持 ID-based Ordering 和 Relaxed Ordering 两种位序。Relaxed Ordering 最早在 PCI-X 总线规范中提出，用来提高 PCI-X 总线的数据传送效率；而 ID-based Ordering 由 PCIe V2.1 总线规范提出。TLP 支持的序如表 6-3 所示。

表 6-3　TLP 支持的序

Attr［2］	Attr［1］	类　　型
0	0	缺省序，即强序模型
0	1	PCI-X Relaxed Ordering 模型
1	0	ID-Based Ordering（IDO）模型
1	1	同时支持 Relaxed Ordering 和 IDO 模型

当使用标准的强序模型时，在数据的整个传送路径中，PCIe 设备在处理相同类型的 TLP 时，如 PCIe 设备发送两个存储器写 TLP 时，后面的存储器写 TLP 必须等待前一个存储器写 TLP 完成后才能被处理，即便当前报文在传送过程中被阻塞，后一个报文也必须等待。

如果使用 Relaxed Ordering 模型，后一个存储器写 TLP 可以穿越前一个存储器写 TLP，提前执行，从而提高了 PCIe 总线的利用率。有时一个 PCIe 设备发出的 TLP，其目的地址并不相同，可能先进入发送队列的 TLP，在某种情况下无法发送，但这并不影响后续 TLP 的发送，因为这两个 TLP 的目的地址并不相同，发送条件也并不相同。

值得注意的是，在使用 PCI 总线强序模型时，不同种类的 TLP 间也可以乱序通过同一条 PCIe 链路，比如存储器写 TLP 可以超越存储器读请求 TLP 提前进行。而 PCIe 总线支持 Relaxed Ordering 模型之后，在 TLP 的传递过程中出现乱序种类更多，但是这些乱序仍然是有条件限制的。在 PCIe 总线规范中为了避免死锁，还规定了不同报文的传送数据规则，即 Ordering Rules。有关 PCIe 总线序的详细说明见第 11 章。

PCIe V2.1 总线规范引入了一种新的“序”模型，即 IDO（ID-Based Ordering）模型，IDO 模型与数据传送的数据流相关，是 PCIe V2.1 规范引入的序模型。有关 PCIe 总线的 Relaxed Ordering 和 IDO 模型的详细说明见第 11.4.2 节。

Attr 字段的第 0 位是“No Snoop Attribute”位。当该位为 0 时表示当前 TLP 所传送的数据在通过 FSB 时，需要与 Cache 保持一致，这种一致性由 FSB 通过总线监听自动完成而不需要软件干预；如果为 1，表示 FSB 并不会将 TLP 中的数据与 Cache 进行一致，在这种情况下，进行数据传送时，必须使用软件保证 Cache 的一致性。

在 PCI 总线中没有与这个“No Snoop Attribute”位对应的概念，因此一个 PCI 设备对存

储器进行 DMA 操作时会进行 Cache 一致性操作[㊀]。这种“自动的”Cache 一致性行为在某些特殊情况下并不能带来更高的效率。

当一个 PCIe 设备对存储器进行 DMA 读操作时，如果传送的数据非常大，比如 512MB，Cache 的一致性操作不但不会提高 DMA 写的效率，反而会降低。因为这个 DMA 读访问的数据在绝大多数情况下，并不会在 Cache 中命中，但是 FSB 依然需要使用 Snoop Phase 进行总线监听。而处理器在进行 Cache 一致性操作时仍然需要占用一定的时钟周期，即在 Snoop Phase 中占用的时钟周期，Snoop Phase 是 FSB 总线事务的一个阶段，如图 3-6 所示。

对于这类情况，一个较好的做法是，首先使用软件指令保证 Cache 与主存储器的一致性，并置“No Snoop Attribute”位为 1[㊁]，然后再进行 DMA 读操作。同理使用这种方法对一段较大的数据区域进行 DMA 写时，也可以提高效率。

除此之外，当 PCIe 设备访问的存储器，不是“可 Cache 空间”时，也可以通过设置“No Snoop Attribute”位，避免 FSB 的 Cache 共享一致性操作，从而提高 FSB 的效率。“No Snoop Attribute”位是 PCIe 总线针对 PCI 总线的不足作出的重要改动。

6.1.4 通用 TLP 头中的其他字段

除了 Fmt 和 Type 字段外，通用 TLP 头还含有以下字段。

1. TH 位、TD 位和 EP 位

TH 位为 1 表示当前 TLP 中含有 TPH（TLP Processing Hint）信息，TPH 是 PCIe V2.1 总线规范引入的一个重要功能。TLP 的发送端可以使用 TPH 信息，通知接收端即将访问数据的特性，以便接收端合理地预读和管理数据，TPH 的详细介绍见第 6.3.6 节。

TD 位表示 TLP 中的 TLP Digest 是否有效，为 1 表示有效，为 0 表示无效。而 EP 位表示当前 TLP 中的数据是否有效，为 1 表示无效，为 0 表示有效。

2. AT 字段

AT 字段与 PCIe 总线的地址转换相关。在一些 PCIe 设备中设置了 ATC（Address Translation Cache）部件，这个部件的主要功能是进行地址转换。只有在支持 IOMMU 技术的处理器系统中，PCIe 设备才能使用该字段。

AT 字段可以用作存储器域与 PCI 总线域之间的地址转换，但是设置这个字段的主要目的是为了方便多个虚拟主机共享同一个 PCIe 设备。对这个字段有兴趣的读者可以参考 Address Translation Sevices 规范，这个规范是 PCI 的 IO Virtualization 规范的重要组成部分。对虚拟化技术有兴趣的读者可以参考清华大学出版社的《系统虚拟化——原理与实现》，以获得基本的关于虚拟化的入门知识。该书主要针对处理器系统的虚拟化技术。而本书将在第 13.2.1 节详细介绍 AT 字段和 PCI 总线相关的虚拟化技术。

3. Length 字段

Length 字段用来描述 TLP 的有效负载（Data Payload）大小[㊂]。PCIe 总线规范规定一个 TLP 的 Data Payload 的大小在 0 ~ 4096 B 之间。PCIe 总线设置 Length 字段的目的是提高总

㊀ PowerPC 处理器通过设置 Inbound 寄存器，也可以避免这个 Cache 一致性操作。

㊁ FSB 收到这类 TLP 后，不进行 Cache 一致性操作。

㊂ 存储器读请求 TLP 没有 DataPayload 字段，此时该 TLP 使用 Length 字段表示需要读取多少数据。

线的传送效率。

当 PCI 设备在进行数据传送时，其目标设备并不知道实际的数据传送大小，这在一定程度上影响了 PCI 总线的数据传送效率。而在 PCIe 总线中，目标设备可以通过 Length 字段提前获知源设备需要发送或者请求的数据长度，从而合理地管理接收缓冲，并根据实际情况进行 Cache 一致性操作。

当 PCI 设备进行 DMA 写操作，将 PCI 设备中 4 KB 大小的数据传送到主存储器时，这个 PCI 设备的 DMA 控制器将存放传送的目的地址和传送大小，然后启动 DMA 写操作，将数据写入到主存储器。由于 PCI 总线是一条共享总线，因此传送 4 KB 大小的数据，可能会使用若干个 PCI 总线写事务才能完成[㊀]，而每一个 PCI 总线写事务都不知道 DMA 控制器何时才能将数据传送完毕。

如果这些总线写事务还通过一系列 PCI 桥才能到达存储器，在这个路径上的每一个 PCI 桥也无法预知这个 DMA 操作何时才能结束，那么这种“不可预知”将导致 PCI 总线的带宽不能被充分利用，而且极易造成 PCI 桥数据缓冲的浪费。

而 PCIe 总线通过 TLP 的 Length 字段，可以有效避免 PCIe 链路带宽的浪费。值得注意的是，Length 字段以 DW 为单位，其最小单位为 1 个 DW。如果 PCIe 主设备传送的单位小于 1 个 DW 或者传送的数据并不以 DW 对界时，需要使用字节使能字段，即“DW BE”字段。有关“DW BE”字段的详细说明见第 6.3.1 节。

6.2 TLP 的路由

TLP 的路由是指 TLP 通过 Switch 或者 PCIe 桥片时采用哪条路径，最终到达 EP 或者 RC 的方法。PCIe 总线一共定义了三种路由方法，分别是基于地址（Address）的路由，基于 ID 的路由和隐式路由（Implicit）方式。

存储器和 I/O 读写请求 TLP 使用基于地址的路由方式，这种方式使用 TLP 中的 Address 字段进行路由选径，最终到达目的地。

而配置读写报文、“Vendor_Defined Messages”报文、Cpl 和 CplD 报文使用基于 ID 的路由方式，这种方式使用 PCI 总线号[㊁]（Bus Number）进行路由选径。在 Switch 或者多端口 RC 的虚拟 PCI-to-PCI 桥配置空间中，包含如何使用 PCI 总线号进行路由选径的信息。

而隐式路由方式主要用于 Message 报文的传递。在 PCIe 总线中定义了一系列消息报文，包括“INTx Interrupt Signaling”，“Power Management Messages”和“Error Signal Messages”等报文。在这些报文中，除了“Vendor_Defined Messages”报文，其他所有消息报文都使用隐式路由方式，隐式路由方式是指从下游端口到上游端口进行数据传递的使用路由方式，或者用于 RC 向 EP 发出广播报文。

6.2.1 基于地址的路由

在 PCIe 总线中，存储器读写和 I/O 读写 TLP 使用基于地址的路由方式。PCIe 设备使用

㊀ 当多个 PCI 设备共享一条 PCI 总线时，一个设备不会长时间占用 PCI 总线，这个设备在使用这条 PCI 总线一定的时间后，将让出 PCI 总线的使用权。

㊁ PCIe 总线实际上使用 Transaction ID 进行 ID 路由，有关 Transaction ID 的详细说明见第 6.3.1 节。

的地址路由方式与 PCI 设备使用的地址路由方式类似。只是 PCIe 设备使用 TLP 进行数据传送，而 PCI 设备使用总线周期进行数据传送。使用地址路由方式进行数据传递的 TLP 格式如第 6.3.1 节的图 6-8 所示，在这类 TLP 中包含目的设备的地址。

当一个 TLP 进行数据传递时，可能会经过多级 Switch，最终到达目的地。Switch 将根据存储器读写和 I/O 读写请求 TLP 的目的地址将报文传递到合适的 Egress 端口上。如图 4-10 所示，在一个 Switch 中包含了多个虚拟 PCI-to-PCI 桥。在 Switch 中有几个端口，就包含几个虚拟 PCI-to-PCI 桥。

在虚拟 PCI-to-PCI 桥的配置寄存器空间中，包含一个桥片能够接收的物理地址范围。PCIe 总线通过这个物理地址范围实现基于地址的路由。这段配置寄存器如图 6-4 所示。当系统软件初始化 PCI 总线时，将合理地设置这些寄存器，之后当 TLP 通过这些 Switch 时将根据这些寄存器选择合适的路径。

Secondary Status	I/O Limit	I/O Base	1Ch
Memory Limit	Memory Base		20h
Prefetchable Memory Limit	Prefetchable Memory Base		24h
Prefetchable Base Upper 32 Bits			28h
Prefetchable Limit Upper 32 Bits			2Ch
I/O Limit Upper 16 Bits	I/O Base Upper 16 Bits		30h

图 6-4　与地址路由相关的 PCI 桥片配置寄存器

图 6-4 中的配置寄存器描述了该虚拟 PCI-to-PCI 桥下游 PCI 子树使用的三组空间范围，分别为 I/O、存储器和可预取的存储器空间，分别用 Base 和 Limit 两类寄存器描述，其中 Base 寄存器表示可访问空间的基地址，Limit 寄存器表示可访问空间的大小。TLP 使用基于地址的路由时，一定要通过查询这组寄存器之后，再决定传送路径。这组寄存器的使用方法与 PCI 总线中的 PCI 桥兼容。

其中 TLP“从上游端口发送到下游端口”与“从下游端口发送到上游端口”的路由过程略有不同，如图 6-5 所示。下文以 TLP1 ~ TLP3 的发送过程对地址路由过程进行说明。TLP1 ~ TLP3 的描述如下。

- TLP1 是一个存储器或者 I/O 请求 TLP，由 RC 发出，并通过一个 Switch 发向 EP1。存储器和 I/O 读写请求 TLP 使用这种地址路由方式。TLP1 将从 Switch 的上游端口传送到下游端口。
- TLP2 是一个存储器或者 I/O 请求 TLP，由 EP2 发出，并通过一个 Switch 发向 RC。当 PCIe 设备进行 DMA 读写操作时，将使用这种地址路由方式。TLP2 将从 Switch 的下游端口传送到上游端口。
- TLP3 是一个存储器或者 I/O 请求 TLP，由一个 EP2 发出，并通过一个 Switch 后发送到另外一个 EP。在 x86 处理器系统中，这种用法并不常见。但是在某些大规模处理器系统中，具有这种应用方式。此时 TLP3 将从 Switch 的下游端口传送到另外一个下游端口。

1. TLP1 的传送过程

当 TLP1 从 RC 发向 EP1 时，这个 TLP1 为 I/O 或者存储器报文，其中 TLP1 目的地址在 EP1 的 BAR 空间中。当处理器访问 EP 的 BAR 空间时，需要使用该类 TLP。值得注意的是

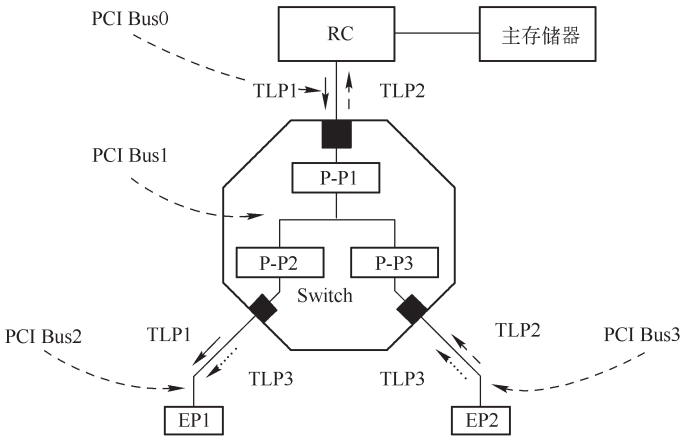

图 6-5　基于地址的路由寻径方式

这个数据报文在通过 RC 时需要进行地址转换。

TLP1 首先通过 PCI Bus0 发向 Switch，并通过 Switch 的 Upstream 端口到达 P-P1 桥片，P-P1 桥片首先根据配置寄存器中的 Limit 和 Base 寄存器决定是否接收 TLP1。如果 Switch 不接收 TLP1，则将该 TLP 作为不支持的请求（Unsupported Request）处理，此时如果 TLP1 需要回应报文，Switch 将发出完成报文，该报文的状态为 UR（Unsupported Request）。

如果 Switch 接收 TLP1，则表示 TLP1 所访问的地址在该 Switch 下游端口所连接的 EP 或者 Switch 中，此时 Switch 将 TLP1 从 PCI Bus0 推至 PCI Bus1 中，即穿越 P-P1 桥片。TLP1 到达 PCI Bus1 后将同时查找 P-P2 和 P-P3 桥片配置寄存器中的 Limit 和 Base 寄存器，决定是 P-P2 还是 P-P3 桥片接收 TLP1。本小节中的例子将使用 P-P2 桥片接收 TLP1，并将 TLP1 推至 PCI Bus2，而 PCI Bus2 上的 EP1 将接收 TLP1，完成整个地址路由。

2. TLP2 的传送过程

当 TLP2 从 EP2 发向 RC 时，一般来说该 TLP 将访问处理器系统的主存储器。此时 TLP2 首先将请求发至 P-P3 桥片，在 P-P3 桥片配置寄存器的 Limit 和 Base 寄存器中当然不会包含 TLP2 所访问的地址，此时 P-P3 桥片将 TLP2 推至 PCI Bus1。

TLP“从下游端口向上游端口”与“从 TLP 从上游端口向下游端口”进行传递时，桥片的处理机制有所不同，从上游端口向下游端口传递时，如果桥片配置寄存器的 Limit 和 Base 寄存器包含该 TLP 的访问地址时，桥片将接收此 TLP，否则不接收该 TLP。而从下游端口向上游端口传递时，如果桥片配置寄存器的 Limit 和 Base 寄存器不包含该 TLP 的访问地址时，桥片将接收该 TLP，并将其推至桥片的上游 PCI 总线。值得注意的是，这两种地址译码方式都属于 PCI 总线的正向译码。

当 TLP2 到达 PCI Bus1 时，首先检查在 PCI Bus1 总线上的 P-P2 桥片是否可以接收此 TLP，如果不能接收则 TLP2 通过 P-P1 桥片传递到 PCI Bus0，即到达 RC。

在 MPC8548 处理器中，到达 RC 的 TLP 首先通过 Inbound 寄存器进行地址转换，将 TLP 的 PCI 总线地址转换为处理器的地址，然后访问处理器中相应的存储器空间；对于 x86 处理器而言，MCH 也会完成 PCI 域地址空间到存储器域地址空间的转换，然后访问处理器中相应的存储器空间。

3. TLP3 的传送过程

TLP3 的传递方式与 TLP2 的传递方式有些类似，当 TLP3 传递到 PCI Bus1 时，P-P2 桥片将接收 TLP3，并将 TLP3 传递到 PCI Bus2 上的 EP1 中。由以上叙述可以发现，PCIe 总线中基于地址的路由方式与 PCI 总线上的基于地址的数据传递流程十分相近。TLP3 在 PCI 总线域上进行数据传递，因此不需要进行 PCI 总线域到存储器域的地址转换。

6.2.2 基于 ID 的路由

在 PCIe 总线中，基于 ID 的路由方式主要用于配置读写请求 TLP、Cpl 和 CplD 报文，此外 Vendor_Defined 消息报文也可以使用这种基于 ID 的路由方式。而在 PCI 总线中，只有配置读写周期才使用 ID 进行数据传递。

基于 ID 的路由方式与基于地址的路由方式有较大的不同，基于 ID 路由方式的 TLP 头格式也与基于地址路由方式的头格式不同，其报文格式如图 6-6 所示。

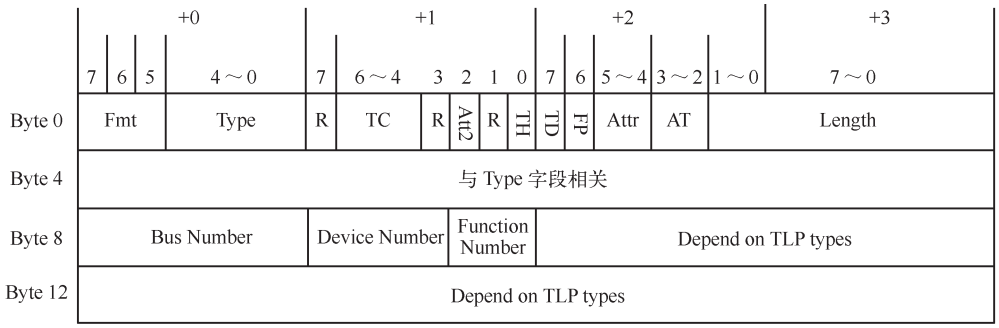

图 6-6 使用 ID 路由的 TLP 头格式

使用 ID 路由方式的 TLP 头，其 Byte8 ~ Byte11 字段与基于地址路由的 TLP 不同。基于 ID 路由的 TLP，使用 Bus Number、Device Number 和 Function Number 进行路由寻址。从软件的角度来看，PCIe 总线与 PCI 总线兼容，只是在 PCIe 总线中，每一个 PCIe 设备使用唯一的 PCI 设备号，但是每一个设备仍然可以有多个子设备（Function）。

PCIe 总线规定，在一个 PCI 总线域空间中，最多只能有 256 条 PCI 总线，因此在一个 TLP 中，Bus Number 由五位组成；而在每一条总线中最多包含 32 个设备，因此 TLP 中的 Device Number 由 5 位[㊀]组成；而每一个设备中最多包含 8 个功能，因此一个 TLP 的 Function Number 由 3 位组成。

配置读写请求 TLP 是使用“基于 ID 路由”的一组重要报文，其主要作用是读写 PCIe 总线的 EP、Switch 及 PCIe 桥片的配置寄存器，以完成 PCIe 总线的配置。在处理器系统上电之后需要进行 PCI 总线系统的枚举，为 PCI 总线分配总线号，并设置 Switch、PCIe 桥片或者 EP 的配置寄存器，如 Limit 寄存器组、Base 寄存器组、BAR 寄存器、Subordinate Bus Number、Secondary Bus Number 和 Primary Bus Number 等一系列配置寄存器。

在上文中我们简单介绍了 Limit 寄存器组和 Base 寄存器组的用法，下文将重点描述 Sub-

㊀ PCIe 链路采用端到端的通信方式，每一个链路只能挂接一个设备，因此在多数情况下，使用 3 位描述 Device Number 是多余的，因此 PCIe 总线提出了 ARI 格式，该格式的详细描述见第 6.3.1 节。

ordinate Bus Number、Secondary Bus Number 和 Primary Bus Number 寄存器。Subordinate Bus Number、Secondary Bus Number 和 Primary Bus Number 寄存器在 Type 01h 配置寄存器中，用来描述 PCI-to-PCI 桥片的上游及下游总线号。这段寄存器在 PCI 配置寄存器中的位置如所图 6-7 示。

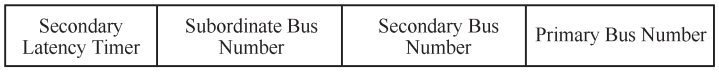

图 6-7　与 ID 路由相关的 PCI 桥片配置寄存器

与 PCI 总线中的桥片类似，Primary Bus Number 记录 PCI-to-PCI 桥上游的 PCI 总线号，Secondary Bus Number 记录 PCI-to-PCI 桥下游的第一个 PCI 总线号，而 Subordinate Bus Number 记录 PCI-to-PCI 桥下游的最后一个 PCI 总线号。

如图 6-5 所示，P-P1 桥片的 Primary Bus Number 为 0，Secondary Bus Number 为 1，而 Subordinate Bus Number 为 3。这些总线号，在处理器系统对 PCI 总线进行枚举时由系统初始化程序设置，从系统初始化程序的角度来看，PCIe 总线与 PCI 总线基本兼容，只是 PCIe 总线对配置空间进行了一些扩展。

在如表 6-2 所示中，RC 可以使用 Type 00h 和 Type 01h 读写请求 TLP，对 PCIe 设备的配置寄存器进行读写访问，配置读写请求 TLP 只能由 RC 发出，配置读写请求 TLP 使用基于 ID 的路由方式。

在如图 6-5 所示的例子中，RC 首先使用 Type 00h 配置请求 TLP 访问在 PCI Bus0 总线上的设备，PCI Bus0 上的所有设备，包括桥片都要监听 PCI Bus 0 上的配置请求，在本例中只有 Switch 挂接在 PCI Bus0 上，实际上是 Switch 的上游端口与 PCI Bus0 直接相连。因此 Switch 的上游端口将接收 RC 发出的 Type 00h 配置请求 TLP，之后 Switch 将向 RC 发出完成报文，结束配置请求。与 PCI 总线相同，PCIe 总线的 Type 00h 类型配置请求 TLP 不能够穿越桥片，在图 6-5 中这类请求只能访问 Switch 上游端口的配置空间。

PCI 总线是基于共享总线的数据传送方式，在一条 PCI 总线上可以连接多个 PCI Agent 设备，其中每一个 PCI Agent 都提供了一个 IDSEL#信号，这个信号与 PCI-to-PCI 桥片或者 HOST 主桥的地址线直接相连，PCI 总线根据与 IDSEL#信号与地址线的连接关系决定相应设备的 Device Number。

这与 PCIe 总线的使用方法不同，PCIe 总线使用“端对端”的连接方式，PCIe 链路只能连接一个下游设备，而这个下游设备的 Device Number 只能为 0。而只有在 Switch 的虚拟 PCI 总线上可以连接多个 Device Number 不同的端口。

当一个虚拟 PCI 总线上挂接 PCI-to-PCI 桥时，系统配置软件将使用 Type 01h 配置请求 TLP 访问 PCI-PCI 桥下游的 PCI 设备。如图 6-5 所示，RC 可以通过 Type 01h 配置请求 TLP 访问 P-P2 桥片、P-P3 桥片，EP1 和 EP2。

当 RC 使用 Type 01h 配置请求 TLP，直接访问 P-P1 桥的下游设备时，首先需要检查该 TLP 的 Bus Number 是否为 1，如果为 1 表示该 TLP 的访问目标在 PCI Bus 1 总线上，此时 PCI-to-PCI 桥将这个 Type 01h 类型的 TLP 转换为 Type 00h 类型的 TLP，然后推至 PCI Bus 1 总线，并访问其下的设备。

如果该 TLP 的 Bus Number 在 P-P1 桥片的 Secondary Bus Number 和 Subordinate Bus Num-

ber 寄存器之间，则 P-P1 桥片将该 Type 01h 类型的 TLP 直接透传到 PCI Bus 1 上，并不改变该 TLP 的类型，之后 Type 01h 类型的 TLP 将继续检查 P-P2 和 P-P3 桥片的配置空间，决定由 P-P2 还是 P-P3 接收该 TLP。如果 TLP 的 PCI Bus Number 为 2 时，P-P2 桥片将接收该 TLP，并将该 Type 01h 类型 TLP 转换为 Type 00h 类型的 TLP，然后发送给 EP1，并由 EP1 处理该 TLP。

上文简要讲述了配置请求 TLP 使用 ID 路由方式从上游端口向下游端口的传递规则，但是 Vendor_Defined 消息报文和 Cpl 和 CplD 报文还可能从下游端口向上游端口进行传递。此时 PCIe 总线处理方法略有不同。下文仍以图 6-5 为例说明这种情况。

当一个 TLP 从 EP2 传送到 EP1 或者 RC 时，首先检查 P-P3 桥片的配置空间，P-P3 桥片发现该 TLP 不是发向自己时，将该 TLP 推至上游总线，即 PCI Bus 1。如果 PCI Bus1 上 P-P1 桥片没有认领该 TLP，该 TLP 将继续向 P-P2 桥片传递，并由这个桥片将 TLP 转发给合适的 EP；如果 P-P1 桥片认领该 TLP，该 TLP 将继续向上游总线传递，直至 RC。

由以上描述可以发现，PCIe 总线使用的基于 ID 的路由方式与 PCI 总线中配置读写总线事务通过 PCI 桥的方法较为类似。

6.2.3 隐式路由

PCIe 总线规定消息请求报文使用隐式路由方式。在 PCIe 总线中，有许多消息是直接发向 RC 或者来自 RC 的广播报文，这些报文不使用地址或者 ID 进行路由，而是使用 Msg 和 MsgD 报文的 Route 字段进行路由，这种路由方式称为隐式路由。

PCIe 总线定义了一些用于中断请求、错误状态处理、锁定总线事务、热插拔信号处理和“Vendor_Defined Messages”消息报文。这些消息报文需要使用隐式路由方式进行传递。消息报文的 Route 字段的含义如表 6-4 所示。

表 6-4 Route［4:0］字段

Route［2:0］	描　　述
000	路由到 RC
001	使用地址路由
010	使用 ID 路由
011	来自 RC 的广播报文
100	本地消息，在接收端结束（Legacy 中断消息使用这种报文格式，传递来自 PCI 总线的中断报文）
101	用于 PCIe 电源管理（PME_TO_Ack 报文使用）
110～111	保留

使用隐式路由方式的 TLP，其 Route 字段为“000”，“011”，“100”或者“101”。当一个报文使用隐式路由向 EP 发送时，EP 将对 Route 字段进行检查，如果这个报文是“来自 RC 的广播报文”，或者是“本地报文”，EP 将接收此报文。

如果 Switch 收到一条使用隐式路由的 TLP 时，将根据报文 Route 字段的不同而分别处理。如果 Switch 的上游端口接收了一条来自 RC 的广播消息，则将该报文发向所有的下游端口；如果 Switch 接收了一条来自下游端口发向 RC 的消息报文时，Switch 将此报文直接转发

到上游端口，直至 RC；如果 Switch 接收了一条使用隐式路由方式的本地消息报文，则 Switch 接收并终结此报文，不再上传或下推。

如果 RC 收到一个使用隐式路由的 TLP 时，将根据报文 Route 字段而分别处理这些 TLP。如果该 Route 字段为 0b000 和 0b101，RC 将接收该 TLP，并作相应的处理；如果为 0b100，RC 将接收该 TLP，并结束该 TLP 报文的传递。

6.3 存储器、I/O 和配置读写请求 TLP

本节讲述 PCIe 总线定义的各类 TLP，并详细介绍这些 TLP 的格式。在这些 TLP 中，有些格式对于初学者来说较难理解。为此本书将在第 12 章中结合一个设计实例，进一步描述这些 TLP 格式。

但是在阅读第 12 章的内容之前，读者需要建立 PCIe 总线中与 TLP 相关的一些基本概念，特别是存储器读写相关的报文格式。在 PCIe 总线中，存储器读写，I/O 读写和配置读写请求 TLP 由以下几类报文组成。

（1）存储器读请求 TLP 和读完成 TLP

当 PCIe 主设备，RC 或者 EP，访问目标设备的存储器空间时，使用 Non-Posted 总线事务向目标设备发出存储器读请求 TLP，目标设备收到这个存储器读请求 TLP 后，使用存储器读完成 TLP，主动向主设备传递数据。当主设备收到目标设备的存储器读完成 TLP 后，将完成一次 DMA 读操作。

（2）存储器写请求 TLP

在 PCIe 总线中，存储器写使用 Posted 总线事务。PCIe 主设备仅使用存储器写请求 TLP 即可完成 DMA 写操作，主设备不需要目标设备的回应报文。

（3）原子操作请求和完成报文

原子操作由 PCIe V2.1 总线规范引入，一个完整的原子操作由原子操作请求和原子操作完成报文组成。原子操作的使用方法与其他 Non-Posted 总线事务类似，首先 PCIe 主设备向目标设备发送原子操作请求，之后目标设备向主设备发送原子操作完成报文，结束一次原子操作。有关原子操作的详细说明见第 6.3.5 节。

（4）I/O 读写请求 TLP 和读写完成 TLP

在 PCIe 总线中，I/O 读写操作使用 Non-Posted 总线事务，I/O 读写 TLP 都需要完成报文做为回应。只是在 I/O 写请求的完成报文中不需要“带数据”，而仅含有 I/O 写请求是否成功的状态信息。

（5）配置读写请求 TLP 和配置读写完成 TLP

从总线事务的角度上看，配置读写请求操作的过程与 I/O 读写操作的过程类似。配置读写请求 TLP 都需要配置读写完成作为应答，从而完成一个完成的配置读写操作。

（6）消息报文

与 PCI 总线相比，PCIe 总线增加了消息请求事务。PCIe 总线使用基于报文的数据传送模式，所有总线事务都是通过报文实现的，PCIe 总线取消了一些在 PCI 总线中存在的边带信号。在 PCIe 总线中，一些由 PCI 总线的边带信号完成的工作，如中断请求和电源管理等，在 PCIe 总线中由消息请求报文实现。

6.3.1 存储器读写请求 TLP

存储器读写请求 TLP 的格式如图 6-8 所示。

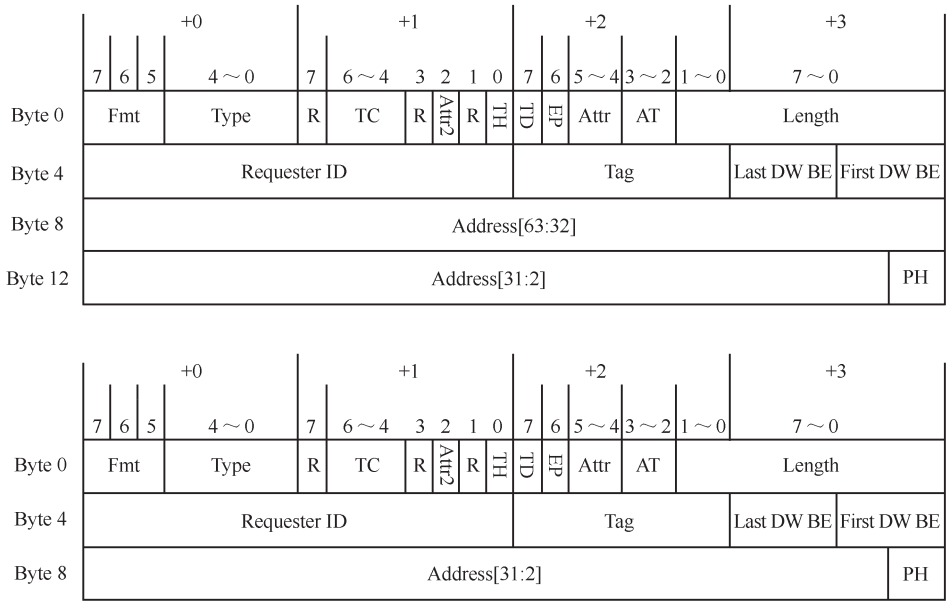

图 6-8　存储器和 I/O 读写请求 TLP 头格式

在 PCIe 总线中，存储器写请求 TLP 使用 Posted 数据传送方式。而其他与存储器和 I/O 相关的报文都使用 Split 方式进行数据传送，这些请求报文需要完成报文，通知发送端之前的数据请求报文已经处理完毕，有关完成报文的详细说明见第 6.3.2 节。

存储器读写请求 TLP 使用地址路由方式进行数据传递，在这类 TLP 头中包含 Address 字段，Address 字段具有两种地址格式，分别是 32 位和 64 位地址。在存储器读写和 I/O 读写请求的第 3 和第 4 个双字中，存放 TLP 的 32 或者 64 位地址。存储器、I/O 和原子操作读写请求使用的 TLP 头较为类似。本节仅介绍存储器、I/O 读写使用的 TLP 头，而在第 6.3.5 节详细介绍原子操作。

1. Length 字段

在存储器读请求 TLP 中并不包含 Data Payload，在该报文中，Length 字段表示需要从目标设备数据区域读取的数据长度；而在存储器写 TLP 中，Length 字段表示当前报文的 Data Payload 长度。

Length 字段的最小单位为 DW。当该字段为 n 时，表示需要获得的数据长度或者当前报文的数据长度为 n 个 DW，其中 $0 \leqslant n \leqslant 0x3FF$。值得注意的是，当 n 等于 0 时，表示数据长度为 1024 个 DW。

2. DW BE 字段

PCIe 总线以字节为基本单位进行数据传递，但是 Length 字段以 DW 为最小单位。为此 TLP 使用 Last DW BE 和 First DW BE 这两个字段进行字节使能，使得在一个 TLP 中，有效数据以字节为单位。

这两个 DW BE 字段各由 4 位组成，其中 Last DW BE 字段的每一位对应数据 Payload 最后一个双字的字节使能位；而 First DW BE 字段的每一位对应数据 Payload 第一个双字的字节使能位。其对应关系如表 6-5 所示。

表 6-5　First 和 Last DW BE 字段

Last DW BE	第 3 位	为 1 表示数据 Payload 的最后一个双字的字节 3 有效
	第 2 位	为 1 表示数据 Payload 的最后一个双字的字节 2 有效
	第 1 位	为 1 表示数据 Payload 的最后一个双字的字节 1 有效
	第 0 位	为 1 表示数据 Payload 的最后一个双字的字节 0 有效
First DW BE	第 3 位	为 1 表示数据 Payload 的第一个双字的字节 3 有效
	第 2 位	为 1 表示数据 Payload 的第一个双字的字节 2 有效
	第 1 位	为 1 表示数据 Payload 的第一个双字的字节 1 有效
	第 0 位	为 1 表示数据 Payload 的第一个双字的字节 0 有效

Last DW BE 和 First DW BE 这两个字段的使用规则如下。

- 如果传送的数据长度在一个对界的双字（DW）之内，则 Last DW BE 字段为 0b0000，而 First DW BE 的对应位置 1；如果数据长度超过 1DW，Last DW BE 字段一定不能为 0b0000。PCIe 总线使用 Last DW BE 字段为 0b0000 表示所传送的数据在一个对界的 DW 之内。
- 如果传送的数据长度超过 1DW，则 First DW BE 字段至少有一个位使能。不能出现 First DW BE 为 0b0000 的情况。
- 如果传送的数据长度大于等于 3DW，则在 First DW BE 和 Last DW BE 字段中不能出现不连续的置 1 位。
- 如果传送的数据长度在 1DW 之内时，在 First DW BE 字段中允许有不连续的置 1 位。此时 PCIe 总线允许在 TLP 中传送 1 个 DW 的第 1，3 字节或者第 0，2 字节。
- 如果传送的数据长度在 2DW 之内时，则 First DW BE 字段和 Last DW BE 字段允许有不连续的置 1 位。

值得注意的是，PCIe 总线支持一种特殊的读操作，即“Zero-Length”读请求。此时 Length 字段的长度为 1DW，而 First DW BE 字段和 Last DW BE 字段都为 0b0000，即所有字节都不使能。此时与这个存储器读请求 TLP 对应的读完成 TLP 中不包含有效数据。再次提醒读者注意“Zero-Length”读请求使用的 Length 字段为 1，而不为 0，为 0 表示需要获得的数据长度为 1024 个 DW。

“Zero-Length”读请求的引入是为了实现“读刷新”操作，该操作的主要目的是为了确保之前使用 Posted 方式所传送的数据，到达最终的目的地，与“Zero-Length”读对应的读完成报文中不含有负载，从而提高了 PCIe 链路的利用率。

在 PCIe 总线中，使用 Posted 方式进行存储器写时，目标设备不需要向主设备发送回应报文，因此主设备并不知道这个存储器写是否已经达到目的地。而主设备可以使用“读刷新”操作，向目标设备进行读操作来保证存储器写最终到达目的地。有关“读刷新”的详细说明及实现原理见第 11 章。

在 PCIe 总线中，标准的存储器读请求也可以完成同样的刷新操作。但是“Zero-Length”读请求与这种读请求相比，其完成报文不需要“Data Payload”，因此在一定程度上提高了 PCIe 总线的效率。如果一个存储器读请求 TLP 报文的 TH 位为 1 时，DW BE 字段将被重新定义为 ST［7:0］字段，有关 ST 字段的详细说明见第 6.3.6 节。

3. Requester ID 字段

Requester ID 字段包含“生成这个 TLP 报文”的 PCIe 设备的总线号（Bus Number）、设备号（Device Number）和功能号（Function Number），其格式如图 6-9 所示。对于存储器写请求 TLP，Requester ID 字段并不是必须的，因为目标设备收到存储器写请求 TLP 后，不需要完成报文作为应答，因此 Requester ID 字段对于存储器写请求 TLP 并没有实际意义。

但是 PCIe 总线规范并没有明确说明存储器写请求 TLP 究竟需不需要 Requester ID 字段，为此 IC 设计者依然需要将存储器写 TLP 的 Requester ID 字段置为有效。值得注意的是，如果一个存储器写请求 TLP 报文的 TH 位为 1 时，Tag 字段将被重新定义为 ST［7:0］字段，有关 ST 字段的详细说明见第 6.3.6 节。

对于 Non-Posted 数据请求，目标设备需要使用完成报文做为回应。在这个完成报文中，需要使用源设备的 Requester ID 字段。因此在 Non-Posted 数据请求 TLP 中，如存储器读请求、I/O 和配置读写请求 TLP，必须使用 Requester ID 字段。

存储器，I/O 读请求 TLP 中含有 Requester ID 和 Tag 字段。在 PCIe 总线中 Requester ID 和 Tag 字段合称为 Transaction ID，Transaction ID 字段的格式如图 6-9 所示。存储器读，I/O 和配置读写请求 TLP 使用 Transaction 字段的主要目的是使接收端通过分析报文的 Transaction ID，确认完成报文的目的地。

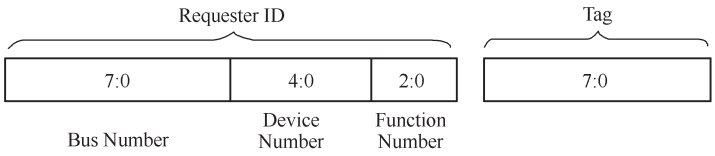

图 6-9　Transaction ID 的格式

在 PCIe 总线中，所有 Non-Posted 数据请求都需要完成报文作为应答，才能结束一次完整的数据传递。一个源设备在发送 Non-Posted 数据请求之后，如果并没有收到目标设备回送的完成报文，TLP 报文的发送端需要保存这个 Non-Posted 数据请求，此时该设备使用的 Transaction ID（Tag 字段）不能被再次使用，直到一次数据传送结束，即数据发送端收齐与该 TLP 对应的所有完成报文。

在同一个时间段内，PCIe 设备发出的每一个 Non-Posted 数据请求 TLP，其 Transaction ID 必须是唯一的。即在同一时间段内，在当前 PCI 总线域中不能存在两个或者两个以上的存储器读请求 TLP，其 Transaction ID 完全相同。

源设备发送 Non-Posted 数据请求后，在没有获得全部完成报文之前，不能释放这个 Transaction ID 占用的资源。在同一个 PCIe 设备发送的 TLP 中，其 Requester ID 字段是相同的，因此 PCIe 设备的设计者所能管理的资源是 Tag 字段。PCIe 设备的设计者需要合理地管理 Tag 资源，以保证数据传送的正确性。

PCIe 设备在发送 Non-Posted 数据请求时，需要暂存这些 Non-Posted 数据请求。其中 Tag

字段的长度决定了发送端能够暂存多少个同类型的 TLP，如果 Tag 字段长度为 5，发送端能够暂存 32 个报文；如果 PCIe 设备使能了 Extended Tag 位（该位的详细描述见第 4. 3. 2 节），Tag 字段可以由 8 位组成，此时发送端能够暂存 256 个报文。

通过 Tag 字段的长度，可以发现每个 PCIe 设备最多可以暂存 256 个同类型的 Non-Posted 报文。但是在多数情况下，一个 PCIe 设备可能只包含 1 个 Function。因此 PCIe 设备还可以使用 Function 号扩展 Tag 字段，从而扩展“暂存 TLP 报文”的数目。

PCIe 设备在 PCI Express Capability 结构的 Device Control 寄存器中，设置了一个 Phantom Functions Enable 位，该位的详细说明见第 4. 3. 2 节。当一个 PCIe 设备仅支持一个 Function 时，Phantom Functions Enable 位可以被设置为 1，此时 PCIe 设备可以使用 Requester ID 的 Function Number 字段对 Tag 字段进一步扩展，此时一个 PCIe 设备最多可以支持 2048 个同类型的数据请求。

由以上分析可以发现，一个 PCIe 设备最多可以支持 2048 个存储器读数据请求，基本上可以满足绝大多数需要。但是在某些特殊应用场合，PCIe 设备即使可以暂存 2048 个存储器读请求，也并不足够。

与 PCI 总线相比，PCIe 总线的数据传送延时较长，而为了弥补这个传送延时，PCIe 设备通常使用流水线技术。此时 PCIe 设备必须能够连续发送多个存储器读请求报文，随后 RC 也将连续回送多个存储器读完成报文，在 PCIe 设备的实现中，需要保证能够源源不断地从 RC 接收这些报文，以充分利用报文接收流水线，有关这部分内容详见第 12. 4. 3 节。

PCIe V2. 1 总线规范还提出了另一种 Requester ID 格式，即 ARI（Alternative Routing-ID Interpretation）格式，除了 Requester ID 外，在完成报文中使用的 Completer ID 也可以使用这种格式。ARI 格式将 ID 号分为两个字段，分别为 Bus 号和 Function 号，而不使用 Device 号，ARI 格式如图 6-10 所示。

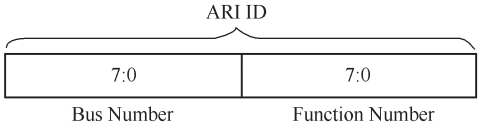

图 6-10　ARI 格式

PCIe 总线引入 ARI 格式的依据是在一个 PCIe 链路上仅可能存在一个 PCIe 设备，因而其 Device 号一定为 0。在多数 PCIe 设备中，Requester ID 和 Completion ID 包含的 Device 号是没有意义的。使用 ARI 格式时，一个 PCIe 设备最多可以支持 256 个 Function，而传统的 PCIe 设备最多只能支持 8 个 Function。

4. I/O 读写请求 TLP 的规则

I/O 读写请求与存储器读写请求 TLP 的格式基本类似，只是 I/O 读写请求 TLP 只能使用 32 位地址模式和基于地址的路由方式，而且 I/O 读写请求 TLP 只能使用 Non-Posted 方式进行传递。PCIe 总线并不建议 PCIe 设备支持 I/O 地址空间，但是 Switch 和 RC 需要具备接收和发送 I/O 请求报文的能力，因为许多老的 PCI 设备依然使用 I/O 地址空间，这些 PCI 设备可以通过 PCIe 桥连接到 PCIe 总线中。因此虽然支持 I/O 读写请求的 PCIe 设备极少，但是在 PCIe 体系结构中，依然需要支持 PCI 总线域的 I/O 地址空间。

与存储器读写请求 TLP 不同，I/O 读写请求 TLP 的某些字段必须为以下值。

- TC［2:0］必须为0，I/O请求报文使用的TC标签只能为0。
- TH和Attr2位保留，而Attr［1:0］必须为“0b00”，这表示I/O请求报文必须使用PCI总线的强序数据传送模式，而且在传送过程中，硬件保证其传送的数据与Cache保持一致，实际上I/O地址空间都是不可Cache的。
- AT［1:0］必须为“0b00”，表示不支持地址转换，因此在虚拟化技术中，并不处理PCI总线域中的I/O空间。
- Length［9:0］为“0b00 0000 0001”，表示I/O读写请求TLP最大的数据Payload为1DW，该类TLP不支持突发传送。
- Last DW［3:0］为“0b0000”。

6.3.2 完成报文

PCIe总线支持Split传送方式，目标设备使用完成报文向源设备主动发送数据。完成报文使用ID路由方式，由TLP Predix、报文头和Data Payload组成，但是某些完成报文可以不含有Data Payload，如I/O或者配置写完成和Zero-Length读完成报文。在PCIe总线中，有以下几类数据请求需要收到完成报文之后，才能完成整个数据传送过程，完成报文格式如图6-11所示。

- 所有的数据读请求，包括存储器、I/O读请求、配置读请求和原子操作请求。当一个PCIe设备发出这些数据请求报文后，必须收到目标设备的完成报文后，才能结束一次数据传送。这一类完成报文必须包含Data Payload。
- 所有的Non-Posted数据写请求，包括I/O和配置写请求。当一个PCIe设备发出这些数据请求报文后，必须收到目标设备的完成报文后，才能结束数据传送。但是这一类完成报文不包含数据，仅包含应答信息。
- 与ATS机制相关的一些报文，详见第13.2节。

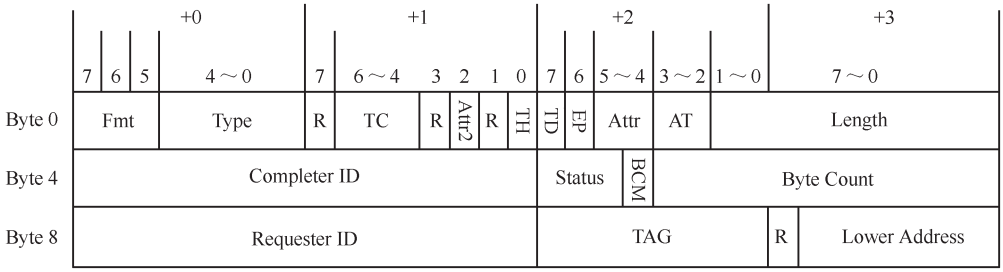

图6-11 完成报文头格式

完成报文“Byte 0”中的大部分字段与“存储器，I/O、配置请求报文”的对应字段的含义相同。完成报文一次最多能够传送的报文大小不能超过Max_Payload_Size参数。在多数处理器中，完成报文中包含的数据在一个Cache行之内，完成报文使用RCB参数来处理数据对界，RCB参数的大小与处理器系统的Cache行长度和DDR-SDRAM的一次突发传送长度相关，这些参数的详细描述见第6.4.3节。在x86和PowerPC处理器中，一个存储器读完成报文一般不超过RCB参数。

1. Requester ID 和 Tag 字段

完成报文使用 ID 路由方式，ID 路由方式详见第 6.2.2 节。完成报文头的长度为 3DW，完成报文头中包含 Transaction ID 信息，由 Requester ID 和 Tag 字段组成，这个 ID 必须与源设备发送的数据请求报文的 Transaction ID 对应，完成报文使用 Transaction ID 进行 ID 路由，并将数据发送给源设备。

当 PCIe 设备收到存储器读、I/O 读写或者配置读写请求 TLP 时，需要首先保存这些报文的 Transaction ID，之后当该设备准备好完成报文后，将完成报文的 Requester ID 和 Tag 字段赋值为之前保存的 Transaction ID 字段。

2. Completer ID 字段

Completer ID 字段的含义与 Requester ID 字段较为相似，只是该字段存放“发送完成报文”的 PCIe 设备的 ID 号。PCIe 设备进行数据请求时需要在 TLP 字段中包含 Requester ID 字段；而使用完成报文结束数据请求时，需要提供 Completer ID 字段。

3. Status 字段

Status 字段保存当前完成报文的完成状态，表示当前 TLP 是正确地将数据传递给数据请求端；还是在数据传递过程中出现错误；或者要求数据请求方进行重试。PCIe 总线规定了几类完成状态，如表 6-6 所示。

表 6-6　Status 字段

Status［2:0］	描　　述
0b000	SC（Sucessful Completion），正常结束
0b001	UR（Unsupported Request），不支持的数据请求
0b010	CRS（Configuration Request Retry Status），要求数据请求方进行重试。当 RC 对一个 PCIe 目标设备发起配置请求时，如果该目标设备没有准备好，可以向 RC 发出 CRS 完成报文，当 RC 收到这类报文时，不能结束本次配置请求，必须择时重新发送配置请求
0b100	CA（Completion Abort），数据夭折。表示目标设备无法完成本次数据请求
其他	保留

4. BCM 位与 Byte Count 字段

BCM（Byte Count Modified）字段由 PCI-X 设备设置。PCI-X 设备也支持 Split Transaction 传送方式，当 PCI-X 设备进行存储器读请求时，目标设备不一定一次就能将所有数据传递给源设备。此时目标设备在进行第一次数据传送时，需要设置 Byte Count 字段和 BCM 位。

BCM 位表示 Byte Count 字段是否被更改，该位仅对 PCI-X 设备有效，而 PCIe 设备不能操纵 BCM 位，只有 PCI-X 设备或者 PCIe-to-PCI-X 桥可以改变该位。本节对此位不做进一步介绍，对此位感兴趣的读者可以参考 PCI-X Addendum to the PCI Local Bus Specification，Revision 2.0。

Byte Count 字段记录源设备还需要从目标设备中获得多少字节的数据就能完成全部数据传递，当前 TLP 中的有效负载也被 Byte Count 字段统计在内。该字段由 12 位组成。该字段为 0b0000-0000-0001 表示还剩一个字节，为 0b1111-1111-1111 表示还剩 4095 个字节，而为 0b0000-0000-0000 表示还剩 4096 个字节。除了存储器读请求的完成报文外，大多数完成报

文的 Byte Count 字段为 4。

如一个源设备向目标设备发送一个“读取 128B 的存储器读请求 TLP”，而目标设备收到这个读请求 TLP 后，可能使用两个存储器读完成 TLP 传递数据。其中第 1 个存储器读完成 TLP 的有效数据为 64 B，而 Byte Count 字段为 128；第 2 个存储器读完成 TLP 中的有效数据为 64 B，而 Byte Count 字段也为 64。当数据请求端接收完毕第 1 个存储器读完成 TLP 后，发现还有 64 B 的数据没有接收完毕，此时必须等待下一个存储器读完成 TLP。等到数据请求端收齐所有数据后，才能结束整个存储器读请求。

目标设备发出的第 2 个读完成 TLP 中的有效数据为 64B，而 Byte Count 字段为 64，当数据请求端接收完毕这个读完成 TLP 后，将完成一个完整的存储器读过程，从而可以释放这个存储器读过程使用的 Tag 资源。

存储器读请求的完成报文的拆分方式较为复杂，Byte Count 字段的设置也相对较为复杂。在第 12 章将结合一个实例讲述该字段的使用方法。

5. Lower Address 字段

如果当前完成报文为存储器读完成 TLP，该字段存放在存储器读完成 TLP 中第一个数据所对应地址的最低位。值得注意的是，在完成报文中，并不存在 First DW BE 和 Last DW BE 字段，因此接收端必须使用存储器读完成 TLP 的 Low Address 字段，识别一个 TLP 中包含数据的起始地址。第 12. 2. 2 节将详细介绍该字段的作用。

6. 3. 3 配置读写请求 TLP

配置读写请求 TLP 由 RC 发起，用来访问 PCIe 设备的配置空间。配置请求报文使用基于 ID 的路由方式。PCIe 总线也支持两种配置请求报文，分别为 Type 00h 和 Type 01h 配置请求。配置请求 TLP 的格式如图 6-12 所示。

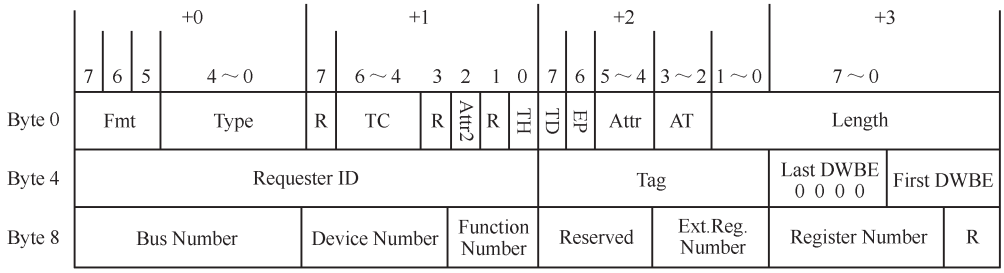

图 6-12 配置请求报文头格式

配置请求 TLP 的第 4 ~ 7 字节与存储器请求 TLP 类似。而第 8 ~ 11 字节的 Bus、Device 和 Function Number 中存放该 TLP 访问的目标设备的相应的号码，而 Ext Register 和 Reigister Number 存放寄存器号。配置请求报文的其他字段必须为以下值。

- TC［2:0］必须为 0，I/O 请求报文的传送类型（TC）只能为 0。
- TH 位为保留位；Attr2 位为保留，而 Attr［1:0］必须为“00b”，这表示 I/O 请求报文使用 PCI 总线的强序数据传送模式；AT［1:0］必须为“0b00”，表示不进行地址转换。

- Length［9:0］为“0b00 0000 0001”，表示配置读写请求最大 Payload 为 1DW。
- Last DW BE 字段为“0b0000”。而 First DW BE 字段根据配置读写请求的大小设置。

6.3.4 消息请求报文

在 PCIe 总线中，多数消息报文使用隐式路由方式，其格式如图 6-13 所示。其中 Byte 0 字段为通用 TLP 头，而 Byte 4 的第 3 字节中存放 Message Code 字段。

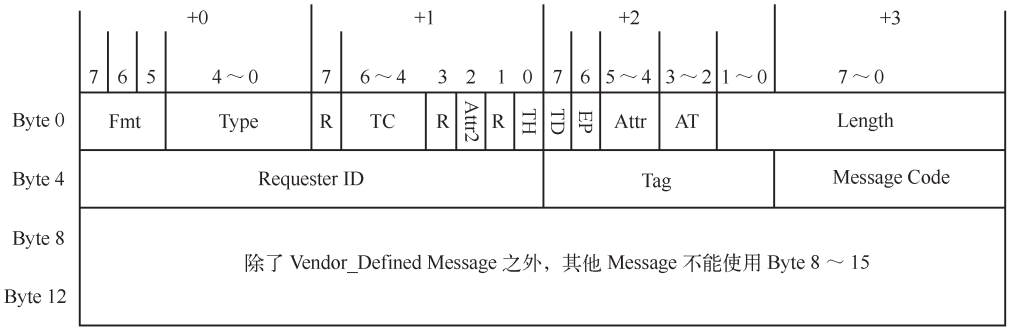

图 6-13　Message 请求 TLP 头格式

PCIe 总线规定了以下几类消息报文。

- INTx 中断消息报文（INTx Interrupt Signaling）。
- 电源管理消息报文（Power Management）。
- 错误消息报文（Error Signaling）。
- 锁定事务消息报文（Locked Transaction Support）。
- 插槽电源限制消息报文（Slot Power Limit Support）。
- Vendor-Defined Messages。

本节将重点讲述 INTx 中断和错误信息相关的消息报文，请读者阅读 PCIe 总线规范了解其他消息报文。

1. INTx 中断消息报文

PCIe 总线推荐设备使用 MSI 或者 MSI-X 机制提交中断请求，但是 MSI 中断机制并不是由 PCIe 总线首先提出的，在 PCI 总线中就已经存在这种中断请求机制。

在 PCI 总线中，虽然提出了 MSI 中断机制，但是几乎没有 PCI 设备使用这种机制进行中断请求。MSI 中断机制是一种基于存储器写的中断请求机制，而 PCI 设备提交 MSI 中断请求，将占用 PCI 总线的带宽，因此多数 PCI 设备使用 INTx 信号进行中断请求。

在 PCIe 总线中，PCIe 设备可以使用 Legacy 中断方式提交中断请求，此时需要使用 INTx 中断消息报文向 RC 通知中断事件。除此之外在 PCIe 体系结构中仍然存在 PCI 设备，这些设备可能使用 INTx 信号提交中断请求。

例如在 PCIe 桥片上挂接的 PCI 设备可能并不支持 MSI 中断机制，因此需要使用 INTx 中断信号提交中断请求，此时 PCIe 桥需要将 INTx 信号转换为 INTx 中断消息报文，并向 RC 提交中断请求。在 PCIe 总线中，共有 8 种 INTx 中断消息报文，见表 6-7。

表 6-7　INTx 中断消息报文

名　　称	Code［7:0］	Routing r［2:0］	Requester ID	描　　述
Assert_INTA	0010 0000	100	包括总线号和设备号，功能号保留	置 INTA#信号有效。注意在 PCIe 总线中，并没有物理上的 INTA#信号
Assert_INTB	0010 0001	100	同上	置 INTB#信号有效
Assert_INTC	0010 0010	100	同上	置 INTC#信号有效
Assert_INTD	0010 0011	100	同上	置 INTD#信号有效
Deassert_INTA	0010 0100	100	同上	置 INTA#信号无效
Deassert_INTB	0010 0101	100	同上	置 INTB#信号无效
Deassert_INTC	0010 0110	100	同上	置 INTC#信号无效
Deassert_INTD	0010 0111	100	同上	置 INTD#信号无效

当 PCIe 设备不使用 MSI 报文向 RC 提交中断请求时，可以首先使用 Assert_INTx 报文向处理器系统提交中断请求，当中断处理完毕，再使用 Deassert_INTx 报文。这些 INTx 中断消息报文的 r［2:0］字段为 0b100，即为 Local 消息报文。设备收到该消息报文后，将结束收到的 INTx 中断消息报文，然后产生一个新的 INTx 中断消息报文。

在一个处理器系统中，PCI 设备首先需要通过 PCIe 桥，之后可能通过多级 Switch，最终到达 RC。假设 PCI 设备使用 INTA#信号进行中断请求，但是由于中断路由表的存在，PCIe 桥可能将 INTA#信号转换为 INTB 中断消息，而这个 INTB 中断消息通过 Switch 时，可能又被 Switch 的中断路由表转换为 INTC 中断消息。因此 PCIe 设备收到 INTx 中断消息后，首先需要结束当前中断消息，之后根据中断路由表产生一个新的 INTx 中断消息，直到这个中断消息传递到 RC。

2. 错误消息报文

在第 4.3.3 节中简要介绍了 AER Capability 结构。如果 PCIe 设备支持 AER Capability 结构，当 PCIe 设备出现某种错误时，将向 RC 或者 RC Event Collector 发送错误消息报文，之后 RC 或者 RC Event Collector 将根据错误类型分别进行处理。

PCIe 总线规范定义了两大类错误类型，分别是可恢复错误（Correctable Errors）和不可恢复错误（Uncorrectable Errors），不可恢复错误又细分为致命错误（Fatal）和非致命错误（Nonfatal）。当 PCIe 设备出现这些错误时，将使用 ERR_COR、ERR_NONFATAL 和 ERR_FATAL 错误消息报文向 RC 或者 RC Event Collector 发送错误消息报文。

PCIe 总线规范并没有详细描述“可恢复错误”和“不可恢复错误”的具体处理方法，也没有详细描述 PCIe 设备的错误恢复机制。对于 PCIe 设备，这些处理方法并不重要。PCIe 总线定义 AER 机制的主要考虑是，由 PCIe 设备将错误信息“统一报告”给 RC 或者 RC Event Collector，并由 RC 或者 RC Event Collector“统一处理”这些错误。其中“统一报告”和“统一处理”才是 AER 机制的设计要点。

当 PCIe 设备出现某种错误后，首先将这些错误信息保留在设备的 AER Capability 结构中，之后 RC 或者 RC Event Collector 从“来自 PCIe 设备的错误信息报文”中获得相应的错误信息。为此在 RC 中设置了一个“Error Source Identification”寄存器保存究竟是哪个 PCIe 设备发出的错误信息报文，之后 RC 或者 RC Event Collector 向处理器系统提交中断请求，由

相应的中断服务例程统一处理所有 PCIe 设备的错误信息。

由 RC 统一处理所有 PCIe 设备错误信息的这种做法，势必加大 RC 和系统软件的设计复杂度。而外部设备的多样性与复杂程度决定了使用这种方法不一定能够取得较好的效果。目前 Intel 的 Chipset 已经支持 AER 机制，但是绝大多数 PCIe 设备并不支持 AER 机制。AER 机制是 Intel 统一外部设备错误处理的一种方法，这为 PCIe 设备的设计提出了更高的要求，而这种方法是否能够取得理想的效果，仍需观察。

6.3.5 PCIe 总线的原子操作

PCIe V2.1 总线规范引入原子操作的概念，原子操作仅能在存储器访问中使用。其中 RC 和 EP 可以作为原子操作的请求者和接收者，而 Switch 和多端口 RC 支持原子操作的转发。PCIe 总线支持三类原子操作，分别是 EP-to-EP，EP-to-RC 和 RC-to-EP 的原子操作。

原子操作有利于提高智能设备[㊀]之间以及智能设备与处理器之间的数据传递效率。当智能设备与处理器进行数据交换时，将不可避免地使用某种锁机制访问临界资源。传统的做法是使用“带锁的”存储器总线事务实现这些锁机制。而使用原子操作可以在很大程度上降低“带锁的”存储器读写请求 TLP 的使用，从而提高 PCIe 总线的使用效率。

PCIe 设备使用一次原子操作可以实现之前需要多次数据操作才能完成的数据交换任务，除此之外 PCIe 设备使用原子操作还可以避免使用带锁的 PCIe 总线事务。原子操作的基本过程如下所示。

（1）源设备向目标设备发送原子操作请求 TLP。原子操作请求 TLP 使用 Non-Posted 方式进行数据传递，且使用基于地址的路由方式。

（2）当目标设备收到这个原子操作请求 TLP 之后，将从这个 TLP 指定的存储器空间中读取原始数据。

（3）目标设备将“原始数据”与“原子操作请求 TLP 中包含的操作数”进行某种规定的运算后产生一个新的数据。这一过程不可被其他总线事务中断，PCIe 设备保证这一过程为原子操作。这个步骤对于原子操作至关重要，也是原子操作的实现要点。

（4）当上述原子操作执行完毕后，目标设备使用原子操作完成报文向源设备传送数据，并将这个新的数据写入目标设备的存储器空间中。原子操作完成报文与存储器读完成的传递方式类似。

由以上分析，可以发现所谓原子操作是指 PCIe 设备“读取原始数据”、“运算”和“产生新的数据”这三个过程不可被其他操作打断。这三个过程将在目标设备中一次完成，并由目标设备的硬件逻辑保证这三个过程不会被其他 TLP 干扰。

由上文所述，一个完整的原子操作由原子操作请求 TLP 和完成 TLP 组成。其中原子操作请求 TLP 的报文头与存储器请求 TLP 类似，如图 6-8 所示。原子操作请求 TLP 具有 Data Payload 字段，在 Data Paylaod 中包含原子操作请求 TLP 使用的操作数。

目前，PCIe 总线共支持 3 种原子操作，分别为 FetchAdd、Swap 和 CAS 原子操作。不同的原子操作使用的操作数个数并不相同，其中 FetchAdd 和 Swap 原子操作使用一个操作数，

㊀ 智能设备中含有一个功能较强的处理器，目前高端显卡、网卡上都包含一个处理器，这类设备都可以被称为智能设备。

而 CAS 原子操作使用两个操作数。

1. FetchAdd 操作

FetchAdd 操作支持 32b 或者 64b 的操作数。如果该 TLP 的 Length 字段为 1DW 时，操作数的长度为 32b；如果该 TLP 的 Length 字段为 2DW 时，操作数的长度为 64b。FetchAdd 操作的执行过程如下所示。

（1）PCIe 设备从 TLP 的指定 PCI 总线地址中获得原始数据。

（2）将原始数据与 TLP 中的操作数相加，并得到一个新的数据。相加的结果忽略进位与溢出位。

（3）将这个新的数据写入 TLP 指定的 PCI 总线地址中。

（4）使用完成报文返回指定 PCI 总线地址中的原始数据。

2. Swap 总线事务

Swap 操作也支持 32b 或者 64b 的操作数，其原则与 FetchAdd 操作完全一致。Swap 操作的执行过程如下所示。

（1）PCIe 设备从 TLP 指定的 PCI 总线地址中读取原始数据。

（2）将 TLP 中的操作数写入 TLP 指定的 PCI 总线地址。

（3）使用完成报文返回 PCI 总线地址中的原始数据。

3. CAS 总线事务

CAS 操作支持 32b、64b 或者 128b 的操作数。如果该 TLP 的 Length 字段为 2DW 时，操作数的长度为 32b；如果该 TLP 的 Length 字段为 4DW 时，操作数的长度为 64b；如果该 TLP 的 Length 字段为 8DW 时，操作数的长度为 128b。CAS 总线事务含有两个操作数，分别为“Compare”和“Swap”。CAS 操作的执行过程如下所示。

（1）PCIe 设备从 TLP 指定的 PCI 总线地址中获得原始数据。

（2）将原始数据与“Compare”操作数进行比较。

（3）如果结果相等，则将“Swap”操作数写入 TLP 指定的位置。

（4）使用完成报文返回 PCI 总线地址中的原始数据。

智能设备之间以及智能设备与处理器之间如果需要使用“Spin Lock”操作时，可以使用 CAS 原子操作实现。值得注意的是，在 x86 处理器的指令集中，也含有 CAS 类指令，该指令是实现“Spin Lock”的基础。但是在处理器系统中使用的“Spin Lock”操作与智能设备使用的“Spin Lock”操作在实现上有所不同。

6.3.6 TLP Processing Hint

当 TLP 的 TH 位为 1 时，表示在当前 TLP 中包含 Processing Hint 字段，PH 字段由 PCIe V2.1 总线规范引入。该字段的引入可以使目标设备根据源设备对数据的使用情况，合理地安排数据缓冲，从而降低 PCIe 设备的访问延时，并最大化地利用 PCIe 设备中的数据缓冲。

Processing Hint 字段的产生与智能设备的大量涌现密切相关。在智能设备中，含有一个运算能力相当强的处理器。智能设备与处理器之间的数据交换，实质上等效于两个处理器系统之间的数据传递。有些智能设备，如在显卡中使用的 GPU（Graphic Processing Unit）和 GP-GPU（Gerneral Purpose GPU）的处理能力甚至超过多数通用处理器。智能设备与处理器系统可以采用图 6-14 所示的拓扑结构连接。

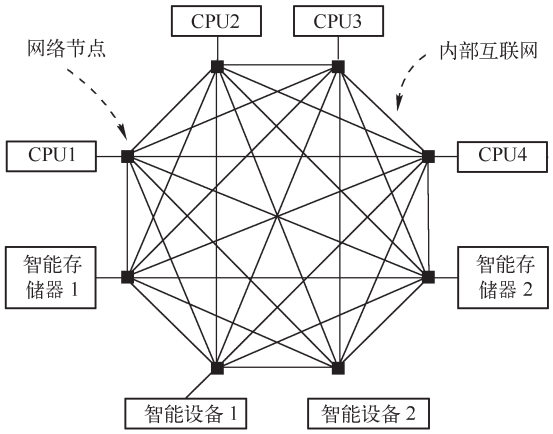

图 6-14　智能外设与处理器系统的连接

在这种处理器系统中，内部互连网络处于核心地位，上图所示的网络是一个理想的全互连结构。在这种互连结构中，处理器、存储器和智能设备与网络节点相连。在这种结构中，在网络节点上连接的设备都含有一个处理器，包括存储器。

当智能设备与处理器进行通信时，如果能够预知数据的使用情况，无疑能够减低数据的访问延时。如智能设备 1 将一组数据写入智能存储器 1 之后，如果这个智能存储器 1 能够预测这组数据将很快被再次使用，则可以将这组数据放入到高速缓冲中，而不必放入低速缓冲中。当这组数据被再次使用时，智能存储器 1 可以很快将数据从高速缓冲中读出，从而缩短了访问延时。

PCIe 总线引入了 PH 字段的主要目的是为了加速外部设备访问主存储器，即 DMA 操作，同时也兼顾 SMP 处理器系统对 PCIe 设备的访问。但是 PH 字段仍然没有完全包含上述模型的所有内容，因为在上述模型中，智能存储器是独立与 PCIe 体系结构的 RC 的，而且智能外设之间具有独立连接通路，在这种模型中，芯片内部的互连网络是真正的设计核心。目前在 P4080 处理器和 Boxboro-EX 处理器系统中，具有全互连网络结构。

1. PH 字段

PCIe 总线使用 PH 字段，使智能设备或者处理器提前预知数据的使用方法。PH 字段仅在与存储器访问相关的 TLP 中出现，该字段由两位组成，在 TLP 中的位置如图 6-8 所示，其详细说明如表 6-8 所示。

表 6-8　PH 字段

PH［1:0］	Processing Hint	描　　述
00	双向数据结构	表示该 TLP 中的数据，经常被源设备和目标设备使用
01	Requester	表示该 TLP 中的数据，经常被源设备使用
10	Target	表示该 TLP 中的数据，经常被目标设备使用
11	Target with Priority	与“Target”类似，但是级别更高

当 PH［1:0］为 0b01 时，表示 TLP 中的数据经常被设备使用，包括以下四种类型。

- DWDW（Device Write after Device Write）。外部设备对一段数据进行写操作后，很快

还会再次进行写操作。

- DWDR（Device Read after Device Write）。外部设备对一段数据进行写操作后，很快还会对这段数据进行读操作。
- DRDW（Device Write after Device Read）。外部设备对一段数据进行读操作后，很快还会对这段数据进行写操作。
- DRDR（Device Read after Device Read）。外部设备对一段数据进行读操作后，很快还会再次进行读操作。

当 PH［1:0］为 0b10 时，表示 TLP 中的数据经常被目标设备使用，包括以下两种类型。在进行 DMA 操作时，HOST 处理器为目标设备，本节也以此为例说明 PH 字段的使用规则。

- DWHR（Host Read after Device Write）。外部设备对一段数据进行写操作后，Host 处理器将很快读取这段数据。
- HWDR（Device Read after Host Write）。Host 处理器对一段数据进行写操作后，外部设备将很快读取这段数据。

2. Steering Tag

通过上文对 PH 字段的描述，可以发现 PH 字段提供的 Processing Hint 控制能力较弱，仅是一个粗颗粒的控制机制。为此 PCIe 总线规范提供了一个 16 位的 ST（Steering Tag）字段，目标设备通过 TLP 中的 Steering Tag 字段可以获得较为详细的信息。

当 TH 位有效时，存储器写请求 TLP 的 Tag 字段被重新定义为 ST［7:0］字段，因为存储器写请求并不需要 Tag 字段；而对于存储器读请求 TLP，TH 位有效时 DW BE 字段被重新定义为 ST［7:0］字段，由于一些存储器读请求 TLP 仍然需要使用 DW BE 字段处理对界，因此 ST 字段只能应用于不需要对界的存储器读请求 TLP。

当 TH 位有效时，这些“不需要对界”的存储器读请求 TLP，将使用默认的 DW BE 值。如果该存储器读请求 TLP 的 Length 字段为 1 时，First DW BE 字段的默认值为 0b1111，而 Last DW BE 字段的默认值为 0b0000；如果 Length 字段不为 1，First 和 Last DW BE 的默认值都为 0b1111。

TLP 还可以支持 ST［15:8］字段，该字段是 ST 的扩展字段。如果一个 TLP 需要使用 ST［15:8］字段，必须使用 TLP Prefix，因为在 TLP 头中已经没有足够的空间放置这些字段。该 TLP Prefix 也被称为 TPH TLP Prefix，其格式如图 6-15 所示。

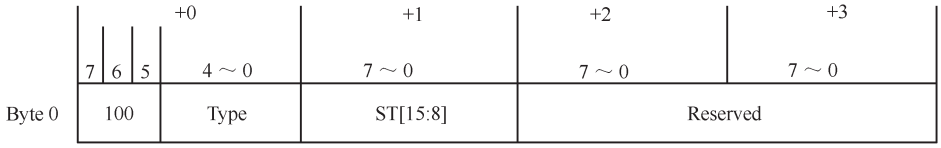

图 6-15　TPH TLP Prefix 格式

如上图所示，在 TPH TLP Prefix 的 Byte 1 中存放 ST［15:8］字段。在 PCIe 总线中，ST［15:8］字段是可选的，实际上整个 ST 字段都是可选的。TPH Requester Capability 结构使用 ST Mode Select 字段定义了 ST 字段的三种使用模式。

- 当该字段为 0b000 时，表示当前 PCIe 设备不支持 ST 字段，此时 TLP 的 ST 字段为

全0。

- 当该字段为0b001时，表示ST模式为“Interrupt Vector Mode”。此时TLP的ST字段由MSI/MSI-X的中断向量号确定。
- 当该字段为0b010时，表示ST字段由PCIe设备决定。

PCIe设备可以根据TLP的属性，决定ST字段的值，为此在一个PCIe设备中，将使用ST表，存放这个设备使用的所有ST字段。这个ST表可以存放在TPH Requester Capability结构中，也可以存放在MSI-X表中。实际上ST表的存放位置并不重要，只要PCIe设备能够根据发出的TLP类型，选择合适的ST字段即可。

目前尚无支持ST字段的PCIe设备，这些PCIe设备发出的TLP中都不包含ST字段。而从PCIe总线规范V2.1中，可以发现MSI/MSI-X中断请求可以使用ST字段，当PCIe设备使用MSI或者MSI-X中断请求时，可以根据中断向量的不同，从ST表中MSI报文[⊖]选择合适的ST字段，然后再发向处理器系统。处理器系统收到这个MSI报文后，可以根据ST字段的不同，分别处理PCIe设备发出的中断请求。

综上所述，ST字段的主要目的是细分TLP的属性，处理器系统可以使用该字段优化数据缓冲，减小数据访问延时。ST字段的支持需要多个PCIe设备共同参与。如EP进行DMA写操作时，数据将发向RC，RC和EP都需要具有解释这个TLP所携带的ST字段的能力。

因此处理器系统在初始化PCIe设备时，需要合理地设置该设备的ST表。目前尚无支持TPH位和ST字段的PCIe设备，但是这种方法可以有效降低PCIe设备访问存储器，以及PCIe设备间数据访问的延时，从而提高PCIe链路的利用率。

6.4 TLP中与数据负载相关的参数

在PCIe总线中，有些TLP含有Data Payload，如存储器写请求、存储器读完成TLP等。在PCIe总线中，TLP含有的Data Payload大小与Max_Payload_Size、Max_Read_Request_Size和RCB参数相关。下面将分别介绍这些参数的使用。

6.4.1 Max_Payload_Size参数

PCIe总线规定在TLP报文中，数据有效负载的最大值为4 KB，但是PCIe设备并不一定能够发送这么大的数据报文。PCIe设备含有“Max_Payload_Size”和“Max_Payload_Size Supported”参数，这两个参数分别在Device Capability寄存器和Device Control寄存器中定义，这两个寄存器在PCI Express Capability结构中的位置见第4.3.2节。

“Max_Payload_Size Supported”参数存放一个PCIe设备中TLP有效负载的最大值，该参数由PCIe设备的硬件逻辑确定，系统软件不能改写该参数。而Max_Payload_Size参数存放PCIe设备实际使用的TLP有效负载的最大值。该参数由PCIe链路两端的设备协商决定，是PCIe设备进行数据传送时实际使用的参数。

PCIe设备发送数据报文时，使用Max_Payload_Size参数决定TLP的最大有效负载。当

⊖ MSI或者MSI-X中断请求使用存储器写请求TLP。

PCIe 设备的所传送的数据大小超过 Max_Payload_Size 参数时，这段数据将被分割为多个 TLP 进行发送。当 PCIe 设备接收 TLP 时，该 TLP 的最大有效负载也不能超过 Max_Payload_Size 参数，如果接收的 TLP，其 Length 字段超过 Max_Payload_Size 参数，该 PCIe 设备将认为该 TLP 非法。

RC 或者 EP 在发送存储器读完成 TLP 时，这个存储器读完成 TLP 的最大 Payload 也不能超过 Max_Payload_Size 参数，如果超过该参数，PCIe 设备需要发送多个读完成报文。值得注意的是，这些读完成报文需要满足 RCB 参数的要求，有关 RCB 参数的详细说明见下文。

在实际应用中，尽管有些 PCIe 设备的 Max_Payload_Size Supported 参数可以为 256 B、512 B、1024 B 或者更高，但是如果 PCIe 链路的对端设备可以支持的 Max_Payload_Size 参数为 128 B 时，系统软件将使用对端设备的 Max_Payload_Size Supported 参数，初始化该设备的 Max_Payload_Size 参数，即选用 PCIe 链路两端最小的 Max_Payload_Size Supported 参数初始化 Max_Payload_Size 参数。

在多数 x86 处理器系统的 MCH 或者 ICH 中，Max_Payload_Size Supported 参数为 128 B。这也意味着在 x86 处理器中，与 MCH 或者 ICH 直接相连的 PCIe 设备进行 DMA 读写时，数据的有效负载不能超过 128 B，同时读完成携带的 Payload 也不能超过 128 B。而在 PowerPC 处理器系统中，该参数大多为 256 B。

目前在大多数 EP 中，Max_Payload_Size Supported 参数不大于 512 B，因为在大多数处理器系统的 RC 中，Max_Payload_Size Supported 参数也不大于 512 B。因此即便 EP 支持较大的 Max_Payload_Size Supported 参数，并不会提高数据传送效率。

而 Max_Payload_Size 参数的大小与 PCIe 链路的传送效率成正比，该参数越大，PCIe 链路带宽的利用率越高，该参数越小，PCIe 链路带宽的利用率越低。

PCIe 总线规范规定，对于实时性要求较高的 PCIe 设备，Max_Payload_Size 参数不应设置过大，因此这个参数有时会低于 PCIe 链路允许使用的最大值。

6.4.2 Max_Read_Request_Size 参数

Max_Read_Request_Size 参数由 PCIe 设备决定，该参数规定了 PCIe 设备一次能从目标设备读取多少数据。

Max_Read_Request_Size 参数在 Device Control 寄存器中定义，详见第 4.3.2 节。该参数与存储器读请求 TLP 的 Length 字段相关，其中 Length 字段不能大于 Max_Read_Request_Size 参数。在存储器读请求 TLP 中，Length 字段表示需要从目标设备读取多少数据。

值得注意的是，Max_Read_Request_Size 参数与 Max_Payload_Size 参数间没有直接联系，Max_Payload_Size 参数仅与存储器写请求和存储器读完成报文相关。

PCIe 总线规定存储器读请求，其读取的数据长度不能超过 Max_Read_Request_Size 参数，即存储器读 TLP 中的 Length 字段不能大于这个参数。如果一次存储器读操作需要读取的数据范围大于 Max_Read_Request_Size 参数时，该 PCIe 设备需要向目标设备发送多个存储器读请求 TLP。

PCIe 总线规定 Max_Read_Request_Size 参数的最大值为 4 KB，但是系统软件需要根据硬

件特性决定该参数的值。因为 PCIe 总线规定 EP 在进行存储器读请求时，需要具有足够大的缓冲接收来自目标设备的数据。

如果一个 EP 的 Max_Read_Request_Size 参数被设置为 4 KB，而且这个 EP 每发出一个 4 KB 大小存储器读请求时，EP 都需要准备一个 4 KB 大小的缓冲[㊀]。这对于绝大多数 EP 都是一个相当苛刻的条件。为此在实际设计中，一个 EP 会对 Max_Read_Request_Size 参数的大小进行限制。

6.4.3 RCB 参数

RCB 位在 Link Control 寄存器中定义，见第 4.3.2 节。RCB 位决定了 RCB 参数的值，在 PCIe 总线中，RCB 参数的大小为 64 B 或者 128 B，如果一个 PCIe 设备没有设置 RCB 的大小[㊁]，则 RC 的 RCB 参数缺省值为 64 B，而其他 PCIe 设备的 RCB 参数的缺省值为 128 B。PCIe 总线规定 RC 的 RCB 参数的值为 64 B 或者 128 B，其他 PCIe 设备的 RCB 参数为 128 B。

在 PCIe 总线中，一个存储器读请求 TLP 可能收到目标设备发出的多个完成报文后，才能完成一次存储器读操作。因为在 PCIe 总线中，一个存储器读请求最多可以请求 4KB 大小的数据报文，而目标设备可能会使用多个存储器读完成 TLP 才能将数据传递完毕。

当一个 EP 向 RC 或者其他 EP 读取数据时，这个 EP 首先向 RC 或者其他 EP 发送存储器读请求 TLP；之后由 RC 或者其他 EP 发送存储器读完成 TLP，将数据传递给这个 EP。

如果存储器读完成报文所传递数据的地址范围没有跨越 RCB 参数的边界，那么数据发送端只能使用一个存储器完成报文将数据传递给请求方，否则可以使用多个存储器读完成 TLP。

假定一个 EP 向地址范围为 0xFFFF-0000 ~ 0xFFFF-0010 的这段区域进行 DMA 读操作，RC 收到这个存储器读请求 TLP 后，将组织存储器读完成 TLP，由于这段区域并没有跨越 RCB 边界，因此 RC 只能使用一个存储器读完成 TLP 完成数据传递。

如果存储器读完成报文所传递数据的地址范围跨越了 RCB 边界，那么数据发送端（目标设备）可以使用一个或者多个完成报文进行数据传递。数据发送端使用多个存储器读完成报文完成数据传递时，需要遵循以下原则。

- 第一个完成报文所传送的数据，其起始地址与要求的起始地址相同。其结束地址或者为要求的结束地址（使用一个完成报文传递所有数据），或者为 RCB 参数的整数倍（使用多个完成报文传递数据）。
- 最后一个完成报文的起始地址或者为要求的起始地址（使用一个完成报文传递所有数据），或者为 RCB 参数的整数倍（使用多个完成报文传递数据）。其结束地址必须为要求的结束地址。
- 中间的完成报文的起始地址和结束地址必须为 RCB 参数的整数倍。

当 RC 或者 EP 需要使用多个存储器读完成报文将 0xFFFE-FFF0 ~ 0xFFFF-00C7 之间的

㊀ 这是流量控制 Infinite FC Unit 的要求，详见第 9.3.2 节。

㊁ 有些 PCIe 设备可能没有 Link Control 寄存器。

数据发送给数据请求方时，可以将这些完成报文按照表 6-9 方式组织。

表 6-9　存储器读完成报文的拆分方法

方式 1	方式 2	方式 3
0xFFFE-FFF0 ~ 0xFFFE-FFFF	0xFFFE-FFF0 ~ 0xFFFE-FFFF	0xFFFE-FFF0 ~ 0xFFFE-FFFF
0xFFFF-0000 ~ 0xFFFF-003F	0xFFFF-0000 ~ 0xFFFF-007F	0xFFFF-0000 ~ 0xFFFF-00C7
0xFFFF-0040 ~ 0xFFFF-007F	0xFFFF-0080 ~ 0xFFFF-00C7	
0xFFFF-0080 ~ 0xFFFF-00BF		
0xFFFF-00C0 ~ 0xFFFF-00C7		

上表提供的方式仅供参考，目标设备还可以使用其他拆分方法发送存储器读完成 TLP。PCIe 总线使用多个完成报文实现一次数据读请求的主要原因是考虑 Cache 行长度和流量控制。在多数 x86 处理器系统中，存储器读完成报文的数据长度为一个 Cache 行，即一次传送 64 B。除此之外，较短的数据完成报文占用流量控制的资源较少，而且可以有效避免数据拥塞。有关流量控制的内容详见第 9 章。

6.5　小结

本章重点介绍 PCIe 总线的事务层。在 PCIe 总线层次结构中，事务层最易理解，同时也与系统软件直接相关。但是事务层的知识较为琐碎，在第 12 章将结合一个 EP 的设计实例，进一步说明 PCIe 总线事务层的具体实现机制。

第 7 章　PCIe 总线的数据链路层与物理层

PCIe 总线的数据链路层处于事务层和物理层之间，主要功能是保证来自事务层的 TLP 在 PCIe 链路中的正确传递，为此数据链路层定义了一系列数据链路层报文，即 DLLP。数据链路层使用了容错和重传机制保证数据传送的完整性与一致性，此外数据链路层还需要对 PCIe 链路进行管理与监控。数据链路层将从物理层中获得报文，并将其传递给事务层；同时接收事务层的报文，并将其转发到物理层。

与事务层不同，数据链路层主要处理端到端的数据传送。在事务层中，源设备与目标设备间的传送距离较长，设备之间可能经过若干级 Switch；而在数据链路层中，源设备与目标设备在一条 PCIe 链路的两端。因此本章在描述数据链路层时，将使用发送端与接收端的概念，而不再使用源设备与目标设备。

物理层是 PCIe 总线的最底层，也是 PCIe 总线体系结构的核心。在物理层中涉及许多与差分信号传递有关的模拟电路知识。PCIe 总线的物理层由逻辑层和电气层组成，其中电气层更为重要。在 PCIe 总线的物理层中，使用 LTSSM 状态机维护 PCIe 链路的正常运转，该状态机的迁移过程较为复杂，在第 8 章将详细介绍该状态机。

7.1　数据链路层的组成结构

数据链路层使用 ACK/NAK 协议发送和接收 TLP，由发送部件和接收部件组成。其中发送部件由 Replay Buffer、ACK/NAK DLLP 接收逻辑和 TLP 发送逻辑组成；而接收部件由“Error Check”逻辑、ACK/NAK 发送逻辑和 TLP 接收逻辑组成。数据链路层的拓扑结构如图 7-1 所示。在该图中含有两个 PCIe 设备，分别为 Device A 和 Device B，使用的 PCIe 链路为 Device A 的发送链路，同时也为 Device B 的接收链路。

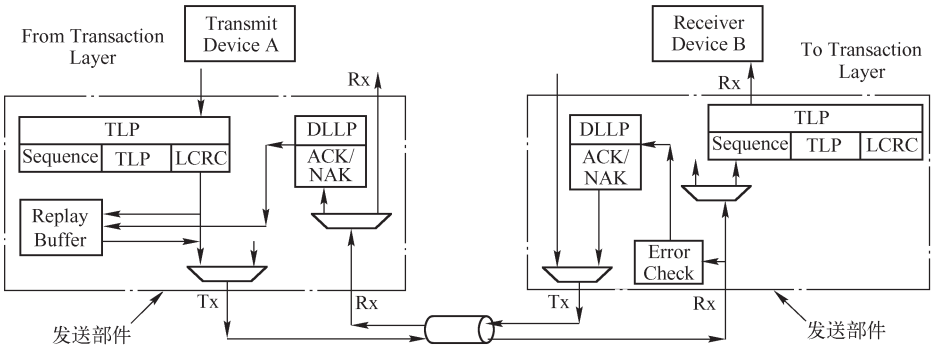

图 7-1　数据链路层的拓扑结构

实际上每个 PCIe 设备的数据链路层都含有发送部件和接收部件。而上图为简化起见，仅含有 Device A 的发送部件和 Device B 的接收部件，即 Device A 发送链路两端使用的两个

部件。Device A 和 Device B 也具有接收部件和发送部件，这两个部件由 Device B 的发送链路使用，Device B 发送链路的工作原理与 Device A 类似，本节对此不做详细介绍。

当 PCIe 设备进行数据传递时，首先在事务层中产生 TLP，然后通过事务层将这个 TLP 发送给数据链路层，数据链路层将这个 TLP 加上 Sequence 前缀和 LCRC 后缀后，首先将这个 TLP 放入到 Replay Buffer 中，然后再发送到物理层。

目标设备（Device B）从物理层接收 TLP 时，将首先获得带前后缀的 TLP，该 TLP 经过数据链路层传递给事务层时，将被去掉 Sequence 前缀和 LCRC 后缀。在数据链路层中，TLP 的格式如图 7-2 所示。

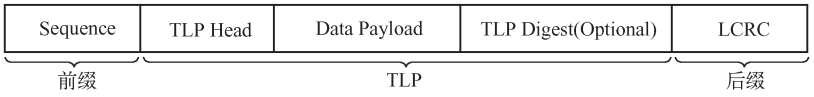

图 7-2　数据链路层 TLP 的格式

数据链路层使用 ACK/NAK 协议保证 TLP 的正确传送，ACK/NAK 协议是一种滑动窗口协议，该协议的详细介绍见第 7.2 节。其中 Sequence 前缀存放当前 TLP 的序列号，滑动窗口协议需要使用这个序列号。该序列号可以循环使用，但在同一个时间段内，一条 PCIe 链路不能含有 Sequence 前缀相同的多个 TLP。而 LCRC 后缀存放当前 TLP 的校验和。

PCIe 总线的数据链路层使用 Replay Buffer[㊀]和 Error Check 部件共同保证数据传送的可靠性和完整性。来自事务层的 TLP 首先暂存在 Replay Buffer 中，然后发送到目标设备。源设备的数据链路层根据来自目标设备的 ACK/NAK DLLP 报文决定是重发这些 TLP，还是清除保存在 Replay Buffer 中的 TLP。

Replay Buffer 的大小决定了事务层可以暂存在数据链路层的报文数，Replay Buffer 的容量越大，在 PCIe 设备发送流水线中容纳的报文越多，从而也容易保证流水线不会因为发送部件出现 underrun 而中断，但是 Replay Buffer 的容量越大，占用的系统资源也越多，从而影响 PCIe 设备的功耗。在一个实际应用中，芯片设计者需要根据 PCIe 链路的延时确定数据链路层 Replay Buffer 的大小，在第 12.4.1 节中将进一步介绍 Replay Buffer 的大小与 PCIe 链路延时间的关系。

在 PCIe 设备的数据链路层中，还含有一个 Error Check 单元。PCIe 设备使用 Error Check 单元检查接收到的 TLP，并决定如何向对端设备进行报文回应。如果 TLP 被正确接收，PCIe 设备将向对端设备发送 ACK DLLP[㊁]；如果 TLP 没有被正确接收，PCIe 设备将向对端设备发送 NAK DLLP。

除了 ACK/NAK DLLP 之外，数据链路层还定义了一系列数据链路层报文 DLLP，以保证 PCIe 链路的正常工作。这些 DLLP 都产生于数据链路层，并终止于数据链路层，并不会传送到事务层。有关 DLLP 格式的详细描述见第 7.1.3 节。

7.1.1　数据链路层的状态

数据链路层需要通过物理层监控 PCIe 链路的状态，并维护数据链路层的“控制与管理

㊀ PCIe 规范将这个 Replay Buffer 称为 Retry Buffer。

㊁ 数据链路层为提高 PCIe 链路的利用率，并不会每成功接收一个 TLP 后，都发送一个 ACK DLLP。

状态机”（Data Link Control and Management State Machine，DLCMSM）。DLCMSM 状态机可以从物理层获得以下与当前 PCIe 链路相关的状态。

- DL_Inactive 状态。物理层通知数据链路层当前 PCIe 链路不可用。在当前 PCIe 链路的对端没有连接任何 PCIe 设备，或者没有检测到对端设备的存在时，数据链路层处于该状态。
- DL_Init 状态。物理层通知数据链路层当前 PCIe 链路可用，且物理层正处于链路初始化状态，此时数据链路层不能接收或者发送 TLP 和 DLLP。此时 PCIe 链路首先需要初始化 VC0 的流量控制机制，然后再对其他虚通路进行流量控制的初始化。有关流量控制的详细描述见第 9 章。
- DL_Active 状态。当前 PCIe 链路处于正常工作模式。此时物理层已完成 PCIe 链路训练或者重训练。有关链路训练的详细描述见第 8 章。

DLCMSM 状态机的迁移模型如图 7-3 所示。

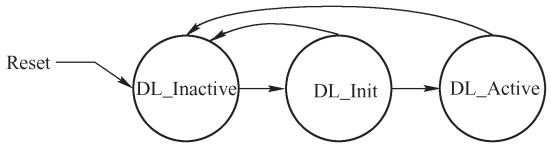

图 7-3　DLCMSM 的状态机模型

DLCMSM 状态机除了可以使用上述状态位，从物理层获得当前 PCIe 链路状态外，还可以使用以下状态位，向事务层通知数据链路层所处的状态。事务层通过这些状态位获知数据链路层所处的工作状态。

- DL_Down。数据链路层处于该状态时，表示在 PCIe 链路的对端没有发现其他设备。当数据链路层处于 DL_Inactive 状态时，该状态位有效。值得注意的是 DL_Down 有效时，并不意味着对端不存在物理设备。数据链路层仅是使用该状态位通知事务层，暂时没有从对端中发现 PCIe 设备，需要进一步检测。
- DL_Up。数据链路层处于该状态表示在 PCIe 链路的对端连接了其他设备。当数据链路层处于 DL_Active 状态时，该状态位有效。

当数据链路层收到物理层的状态信息后，DLCMSM 状态机将进行状态转换，并向事务层通知 PCIe 链路的状态。如果在 PCIe 链路的两端都连接着 PCIe 设备，那么这两个 PCIe 设备的数据链路层，在绝大多数时间内状态相同。数据链路层各个状态的详细说明，及 PCIe 链路的状态迁移过程如下。

1. DL_Inactive 状态

当 PCIe 设备复位时，将进入该状态。值得注意的是，只有传统复位方式才能使 PCIe 设备进入 DL_Inactive 状态，而 FLR 方式并不会影响 DLCMSM 状态机。

当 PCIe 设备从复位状态进入 DL_Inactive 状态时，将对 PCIe 数据链路层进行彻底复位，将与 PCIe 链路相关的寄存器置为复位值，并丢弃在 Replay Buffer 中保存的所有报文。当 PCIe 设备处于 DL_Inactive 状态时，数据链路层将向事务层提交 DL_Down 状态信息，并丢弃来自数据链路层和物理层的所有 TLP，而且不接收对端设备发送的 DLLP。

PCIe 设备的物理层设置了一个 LinkUp 位，该位为 1 时表示 PCIe 链路的对端与一个 PCIe

设备相连。当物理层的 LinkUp 状态位为 1，而且事务层没有禁用当前 PCIe 链路时，PCIe 数据链路层将从 DL_Inactive 状态迁移到 DL_Init 状态。

PCIe 设备在进行链路训练时，将检查 PCIe 链路的对端是否存在 PCIe 设备，如果对端不存在 PCIe 设备，物理层的 LinkUp 位将为 0，此时数据链路层将一直处于 DL_ Inactive 状态。系统软件可以设置 Switch 下游端口 Link Control 寄存器的“Link Disable”位为 1，禁用该端口连接的 PCIe 链路，此时即便 PCIe 链路对端存在 PCIe 设备，数据链路层的状态也仍然为 DL_Inactive。

2. DL_Init 状态

当数据链路层处于 DL_Init 状态时，将对 PCIe 链路的虚通道 VC0 进行流量控制初始化。在 PCIe 总线中，流量控制的初始化分为两个阶段，分别为 FC_INIT1 和 FC_INIT2。在流量控制的 FC_INIT1 阶段，数据链路层将向事务层提交 DL_Down 状态信息；而在流量控制的 FC_INIT2 阶段，数据链路层将向事务层提交 DL_Up 状态信息。流量控制的初始化部分详见第 9.3.3 节。

当 PCIe 链路处于 DL_Down 状态时，发送端可以丢弃任何没有被 ACK/NAK 确认的 TLP，此时数据链路层几乎不会受到事务层的干扰，从而可以保证流量控制初始化的正常进行。这也是 PCIe 链路的流量控制分为 FC_INIT1 和 FC_INIT2 的主要原因。

当 VC0 的流量控制初始化完毕，而且物理层的 LinkUp 状态位为 0b1 时，数据链路层将从 DL_Init 状态迁移到 DL_Active 状态；如果在进行流量控制初始化时，物理层的 LinkUp 状态位被更改为 0b0 时，数据链路层将从 DL_Init 状态迁移到 DL_Inactive 状态。

3. DL_Active 状态

当数据链路层处于 DL_Active 状态时，PCIe 链路可以正常工作，此时数据链路层可以从事务层和物理层正常接收和发送 TLP，并处理 DLLP，此时数据链路层向事务层提交 DL_Up 状态信息。

当发生以下事件后，数据链路层可以从 DL_Active 状态迁移到 DL_Inactive 状态，但是不能迁移到 DL_Init 状态。这也意味着数据链路层从 DL_Active 状态迁移出去后，必须重新进行对端设备的识别和流量控制初始化，之后才能进入 DL_Active 状态。

在多数情况下，数据链路层从 DL_Active 状态迁移到 DL_Inactive 状态时，意味着处理器系统出现了异常，系统软件需要处理这些异常。但是在下列情况时，数据链路层状态从 DL_Active 状态迁移到 DL_Inactive 状态时并不会引发异常。

- Bridge Control Register 的 Secondary Bus Reset 位被系统软件置为 1 时，数据链路层将迁移到 DL_Inactive 状态。
- Link Disable 位被系统软件置为 1 时，数据链路层迁移到 DL_Inactive 状态。
- 当一个 PCIe 端口向对端设备发送“PME_Turn_Off”消息之后，其数据链路层经过一段时间，可以迁移到 DL_Inactive 状态。RC 和 Switch 在进入低功耗状态之前，将向其下游端口广播 PME_Turn_Off 消息，下游 PCIe 设备收到该消息后，将向 RC 和 Switch 发出 PME_TO_Ack 回应。当 RC 和 Switch 的下游端口收到这个回应报文后，数据链路层可以迁移到 DL_Inactive 状态。
- 如果 PCIe 链路连接了一个支持“热插拔”功能的 PCIe 插槽，而当这个插槽的 Slot Capability 寄存器的“Hot Plug Surprise”位为 1 时，数据链路层将迁移到 DL_Inactive

状态。

- 如果 PCIe 链路连接一个热插拔插槽，当这个插槽的 Slot Control 寄存器的“Power Controller Control”位为 1 时，数据链路层也将迁移到 DL_Inactive 状态。

在 PCIe 总线中，还有一些系统事件也可以引发数据链路层的状态转换，本书对此不进行一一描述。

7.1.2 事务层如何处理 DL_Down 和 DL_Up 状态

当事务层收到数据链路层的 DL_Down 状态信息时，表示出现了以下情况。

- PCIe 链路的对端没有连接设备。
- PCIe 链路丢失了与对端设备的连接。
- 数据链路层和物理层出现某种错误，PCIe 链路不能正常工作。
- 系统软件禁用 PCIe 链路。

当事务层收到 DL_Down 状态信息后，将不再从数据链路层中接收 TLP，除非是数据链路层已经使用 ACK/NAK 报文确认过的 TLP。这些被确认过的 TLP 已经被数据链路层接收完毕，因此事务层可以接收这些 TLP。

当链路处于 DL_Down 状态时，RC 或者 Switch 的下游端口将复位与链路相关的内部逻辑和状态。此时下游端口收到上游端口的 Non-Posted 请求 TLP 后，并不会将这个 TLP 转发到数据链路层，因为数据链路层已经出现故障，而组织状态位为 UR（Unsupported Request）的完成报文，通知上游端口无法发送这个 Non-Posted 数据请求，该事务层将丢弃这个 Non-Posted 请求 TLP；此外该事务层还将丢弃来自上游端口的 Posted 请求 TLP 和完成报文。当链路为 DL_Down 状态时，RC 或者 Switch 的下游端口还必须结束“PME Turn_Off”握手请求。

当链路为 DL_Down 状态时，Switch 和 PCIe 桥的上游端口，将复位相关的内部逻辑和状态，并丢弃所有正在处理的 TLP。此时 Switch 和 PCIe 桥将使用 Hot Reset 方式复位所有下游端口。

事务层处于 DL_Up 状态时，表示该设备与 PCIe 链路的对端设备已经建立连接，链路两端可以正常收发报文。当事务层发现 PCIe 链路从 DL_Down 迁移到 DL_Up 状态时，将向 PCIe 链路的对端设备重新发送 Set_Slot_Power_Limit 消息，并重新初始化相关的寄存器。

7.1.3 DLLP 的格式

DLLP 与 TLP 的概念并不相同，DLLP 产生于数据链路层，终止于数据链路层，这些报文不会出现在事务层中，而且对系统软件透明。设置 DLLP 的目的是为了保证 TLP 的正确传送和管理 PCIe 链路。

值得注意的是，DLLP 并不是由 TLP 加上 Sequence 前缀和 LCRC 后缀组成的，而具有单独的格式。一个 DLLP 由 6 个字节组成，其中第 1 个字节存放 DLLP 的类型，第 2 ~ 4 个字节存放的数据与 DLLP 类型相关，而最后两个字节存放当前 DLLP 的 CRC 校验。DLLP 的格式如图 7-4 所示。

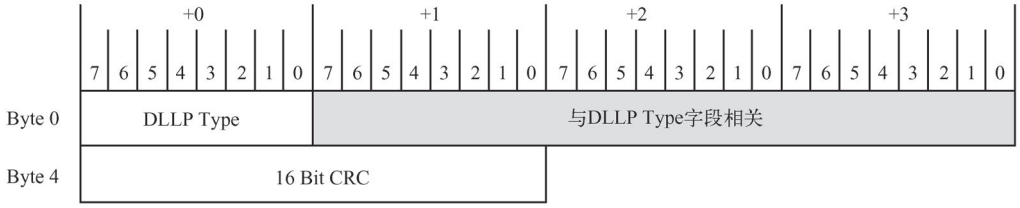

图 7-4　DLLP 的格式

大多数 DLLP 由 PCIe 设备自动产生，而与事务层没有直接联系。PCIe 总线定义了以下几类 DLLP 报文，如表 7-1 所示。

表 7-1　DLLP 的编码

Type 字段	DLLP 类型
0000-0000	Ack
0001-0000	Nak
0010-0000	PM_Enter_L1
0010-0001	PM_Enter_L23
0010-0011	PM_Active_State_Request_L1
0010-0100	PM_Request_Ack
0011-0000	Vendor Specific-Not used in normal operation
0100-0$V_2V_1V_0$	InitFC1-P，其中 V[2:0]由三位组成，因为一条 PCIe 链路最多支持 8 个 VC
0101-0$V_2V_1V_0$	InitFC1-NP
0110-0$V_2V_1V_0$	InitFC1-Cpl
1100-0$V_2V_1V_0$	InitFC2-P
1101-0$V_2V_1V_0$	InitFC2-NP
1110-0$V_2V_1V_0$	InitFC2-Cpl
1000-0$V_2V_1V_0$	UpdateFC-P
1001-0$V_2V_1V_0$	UpdateFC-NP
1010-0$V_2V_1V_0$	UpdateFC-Cpl

这些 DLLP 报文的描述如下。

- ACK DLLP。该 DLLP 由接收端发向发送端。接收端收到 TLP 报文后，将根据数据链路层的阈值设置，向对端设备发送 ACK DLLP，而不是每接收到一个 TLP，都向对端发送一个 ACK DLLP。该 DLLP 表示接收端正确收到来自对端的 TLP。
- NAK DLLP。该 DLLP 由接收端发向发送端。该 DLLP 表示接收端有哪些 TLP 没有被正确接收，发送端收到 NAK DLLP 后，将重传没有被正确接收的 TLP，同时释放已经被正确接收的 TLP。ACK 和 NAK DLLP 与 ACK/NAK 协议相关，是数据链路层的两个重要 DLLP。这两个 DLLP 的详细作用如第 7.2 节所述。
- Power Management DLLPs。PCIe 设备使用该组 DLLP 进行电源管理，并向对端设备通知当前 PCIe 链路的状态。PCIe 总线还定义了一组与电源管理相关的 TLP，这些 TLP

与这组 DLLPs 有一定的联系，但是其作用并不相同。PCIe 总线使用该组 DLLP 保证电源管理状态机的正确运行。

- Flow Control Packet DLLPs。该组 DLLP 包括 InitFC1、InitFC2、UpdateFC DLLP，PCIe 总线使用这些 DLLPs 进行流量控制。在 PCIe 总线中，数据传送由三大类组成，分别为 Posted、Non-Posted 和 Completion。这三种数据传送方式有些细微区别，PCIe 设备为这三种数据传送设置了不同的数据缓冲。流量控制是 PCIe 总线的一个重要特性，第 9 章将重点介绍这些内容。
- Vendor-specific DLLP。一些定制的 DLLP，PCIe 总线规范并未对此约束。这些 DLLP 由用户自定义使用。

本节将重点介绍 ACK/NAK DLLP，这两个 DLLP 与 PCIe 总线的 ACK/NAK 协议直接相关。在 PCIe 总线中，数据链路层使用 ACK/NAK 协议保证 TLP 的可靠传送。ACK/NAK DLLP 的格式如图 7-5 所示。

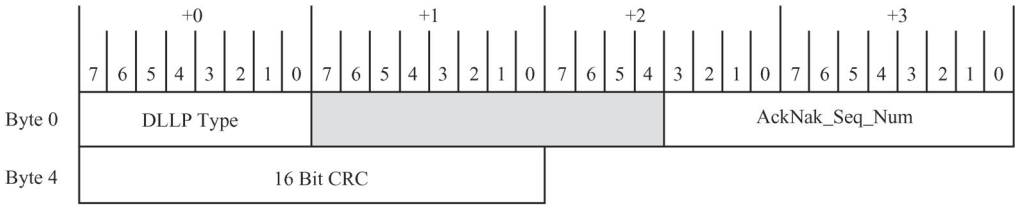

图 7-5　ACK/NAK DLLP

ACK/NAK DLLP 各字段的详细说明如表 7-2 所示。

表 7-2　ACK/NAK DLLP 的详细说明

名　称	位　置	描　述
DLLP Type	Byte 0 的 7 ~ 0 位	0b0000-0000 表示为 ACK 报文 0b0001-0000 表示为 NAK 报文
AckNak_Seq_Num	Byte 2 的 3 ~ 0 位， Byte 3 的 7 ~ 0 位	该字段表示接收端成功接收的报文序号，下文将详细解释该字段。
CRC	Byte 4 ~ Byte 5	保存 DLLP 的 CRC 校验和。

发送端的数据链路层负责将 TLP 传送给接收端，而接收端的数据链路层在收到 TLP 之后，将向发送端发送 ACK/NAK DLLP。发送端和接收端通过某种传送协议，完成数据链路层的数据交换，在 PCIe 总线中，这个协议称为 ACK/NAK 协议。

7.2　ACK/NAK 协议

ACK/NAK 协议是一种滑动窗口协议。PCIe 设备的发送端和接收端分别设置了两个窗口。发送端在发送 TLP 时，首先将这个 TLP 放入发送窗口中（这个窗口即 Replay Buffer），并对这些 TLP 从 0 ~ n 进行编号。只要发送窗口不满，发送端就可以持续地从事务层中接收报文，然后将其放入 Replay Buffer 中。

发送端需要保留在这个窗口中的数据报文，并在收到来自接收端的 ACK/NAK 确认报文

之后，统一释放保存在发送窗口中的报文，并滑动这个发送窗口。当发送端收到接收端对第 n 个报文的确认后，表示第 n、n－1、n－2 等在窗口中的报文都已经被正确收到，然后统一滑动这个窗口。PCIe 总线使用这种方法可以提高窗口的利用率。

与此对应，接收端也维护了一个窗口，该窗口记录数据报文的发送序列号范围。当数据报文到达后，如果其序列号在接收窗口范围内，接收端将接收该报文，并根据根据实际情况，向发送端发送回应报文。这个回应报文包括 ACK 和 NAK DLLP。下文将分别讨论发送端和接收端如何使用 ACK/NAK 协议。

7.2.1 发送端如何使用 ACK/NAK 协议

数据链路层在发送 TLP 之前，发送端首先需要将 TLP 进行封装，加上 Sequence 前缀和 LCRC 后缀，之后再将这个 TLP 放入 Replay Buffer 中。发送端设置了一个 12 位的计数器 NEXT_TRANSMIT_SEQ，这个计数器的初始值为 0，当数据链路层处于 DL_Inactive 状态时，该计数器将保持为 0。为简化起见，本节只讲述数据链路层处于 DL_Active 状态时的情况，而不讲述处于 DL_Inactive 状态时的情况。

发送端使用计数器 NEXT_TRANSMIT_SEQ 的当前值设置 TLP 的 Sequence 号，该计数器的初始值为 0。PCIe 设备每发送完毕一个 TLP，这个计数器将加 1，直到该计数器的值为 4095（NEXT_TRANSMIT_SEQ 的最大值）。当计数器的值为 4095 后，再进行加 1 操作时，该计数器将回归为 0。而 LCRC 是根据 TLP 的内容计算出来的，用来保证数据传递的完整性，本节不介绍 LCRC 的计算过程。对此有兴趣的读者请参考 PCIe 总线规范。

与此对应，接收端也设置了一个 12 位的计数器 NEXT_RCV_SEQ。这个计数器记录接收端即将接收的 TLP 的 Sequence 号。这个计数器的初始值为 0，当数据链路层处于 DL_Inactive 状态时，该计数器保持为 0。在正常情况下，到达接收端的 TLP，其 Sequence 号和这个计数器中的内容一致。当接收端将这个 TLP 转发到事务层后，这个计数器将加 1，当计数器的值为 4095 后，再进行加 1 操作时，该计数器将回归为 0。如果到达接收端的 TLP，其序号与这个计数器中的值不一致时，接收端需要进行特殊处理，详见下文。

发送端为处理来自接收端的 ACK/NAK DLLP，设置了一个 12 位的计数器 ACKD_SEQ。这个计数器记载最近接收到的 ACK/NAK DLLP 的 AckNak_Seq_Num 字段。这个计数器的初始值为全 1，当数据链路层处于 Inactive 状态时，该计数器保持为全 1。发送端收到 ACK/NAK DLLP 后，将使用这些 DLLP 中的 AckNak_Seq_Num 字段更新 ACKD_SEQ 计数器。

如果（NEXT_TRANSMIT_SEQ-ACKD_SEQ）mod 4096 > =2048 时，发送端将不会从事务层继续接收新的 TLP，因为此时发送端已经发送了许多 TLP，但是接收端可能并没有成功接收这些 TLP，因此并没有及时发送 ACK/NAK DLLP 作为回应。在多数情况下，当 PCIe 链路出现了某些问题时，才可能导致该公式成立。此外 ACKD_SEQ 计数器还可以帮助发送端重发错误的 TLP，下文将详细解释这个功能。

发送端首先将从事务层获得的 TLP 存放到 Replay Buffer 中，在 Relpay Buffer 中可以存放多个 TLP，这个 Replay Buffer 为发送端使用的发送窗口。PCIe 总线并没有规定在 Replay Buffer 中存放 TLP 的个数，不同的设计可以采用的大小不同，其中有一个重要的原则就是不能使 Replay Buffer 成为整个设计的瓶颈，Replay Buffer 应该始终保证有足够的空间接收来自事务层的报文。

TLP 进入 Replay Buffer 之后，发送端首先将这个 TLP 封装，然后从 Replay Buffer 中发送到物理层，最终达到接收端。发送端将 TLP 发送出去之后，将等待来自接收端的应答，接收端使用 ACK/NAK DLLP 发送这个应答。发送端根据应答结果决定是将 TLP 从 Replay Buffer 中清除，还是重发在 Replay Buffer 中的 TLP。下文将以几个实例说明发送端如何处理来自接收端的应答。

1. 发送端收到 ACK DLLP 报文

如图 7-6 所示，假设发送端从 Replay Buffer 中向接收端发送 Sequence 号为 3 ~ 7 的报文。接收端收到这些报文后将发送 ACK DLLP 作为回应，其详细步骤如下。

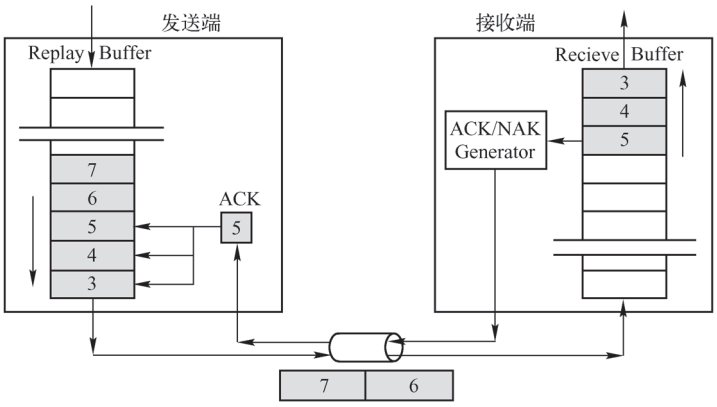

图 7-6　发送端收到 ACK DLLP

（1）发送端向接收端发送 TLP3 ~ 7，其中 TLP3 是第一个报文，TLP7 是最后一个报文。此时发送端的 NEXT_TRANSMIT_SEQ 计数器为 8，表示即将填入到 Replay Buffer 中的报文序列号为 8。

（2）接收端按序收到 TLP3 ~ 5，而 TLP6 和 7 仍在传送过程中。接收端的 NEXT_RCV_SEQ 计数器为 6，表示即将接收的报文序列号为 6。

（3）接收端通过报文检查决定接收 TLP3 ~ 5，然后发送 ACK DLLP，此时这个 ACK DLLP 的 AckNak_Seq_Num 字段为 5。为了提高总线的利用率，接收端不会为每一个接收到的 TLP 都做出应答。在这个例子中，AckNak_Seq_Num 字段为 5 表示 TLP3 ~ 5 都已经被接收。

（4）发送端收到 AckNak_Seq_Num 字段为 5 的 ACK DLLP 后，得知 TLP3 ~ 5 都被成功接收。此时发送端将 TLP3 ~ 5 从 Replay Buffer 中清除。

（5）接收端陆续收到 TLP6 ~ 7 后，接收端的 NEXT_ RCV_ SEQ 计数器为 8，表示即将接收的报文序列号为 8。然后接收端向发送端发送 ACK DLLP，这个 DLLP 的 AckNak_Seq_Num 字段为 7，即为 NEXT_RCV_SEQ - 1。

（6）发送端收到 AckNak_Seq_Num 字段为 7 的 ACK DLLP 后，得知 TLP6 ~ 7 都被成功接收。此时发送端将 TLP6 ~ 7 从 Replay Buffer 中清除。

2. 发送端收到 NAK DLLP 报文

如图 7-7 所示，假设发送端从 Replay Buffer 中向接收端发送 Sequence 号为 3 ~ 7 的报文。接收端收到这些报文后，发现有错误的 TLP，此时将发送 NAK DLLP 而不是 ACK DLLP，

其详细步骤如下。

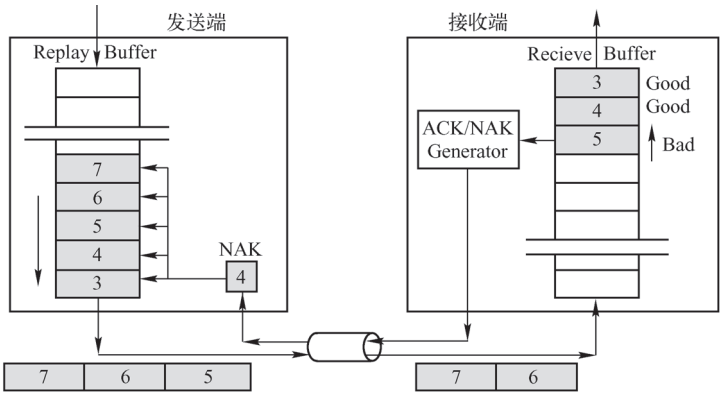

图 7-7　发送端收到 NAK DLLP

（1）发送端向接收端发送 TLP3 ~ 7，其中 TLP3 是第一个报文，而 TLP7 是最后一个报文。

（2）接收端按序收到 TLP3 ~ 5，而 TLP6 和 7 仍在传送过程中。

（3）接收端通过报文检查决定接收 TLP3 ~ 4，此时 NEXT_RCV_SEQ 为 5，表示即将接收 TLP5。

（4）TLP5 没有通过完整性验证，此时接收端将向对端发送 NAK DLLP，这个 DLLP 的 AckNak_Seq_Num 字段为 4，即为 NEXT_RCV_SEQ - 1。AckNak_Seq_Num 字段为 4 表示接收端最后一个接收正确的 TLP，其 Sequence 号为 4。

（5）发送端收到 AckNak_Seq_Num 字段为 4 的 NAK DLLP 后，得知 TLP3 ~ 4 已被成功接收。此时发送端首先停止从事务层接收新的 TLP，之后将 TLP3 ~ 4 从 Replay Buffer 中清除。

（6）发送端重新发送在 Replay Buffer 中从 TLP5 开始的报文。在这个例子中，发送端将重新发送 TLP5 ~ 7。

发送端每一次收到 NAK DLLP 后，都将重发在 Replay Buffer 中剩余的 TLP。但是发送端不能无限次重发同一个 TLP，因为出现这种情况意味着链路出现了某些问题，必须修复这些问题后，才能继续重发这些 TLP。为此在发送端中设置了一个 2 位计数器 REPLAY_NUM，这个计数器的初始值为 0，当数据链路层处于 Inactive 状态时，该计数器保持为 0。

REPLAY_NUM 计数器按照以下几个原则进行更新。

- 当发送端第一次收到 NAK DLLP 后，REPLAY_NUM 计数器将加 1，此时 ACKD_SEQ 计数器被赋值为这个 NAK DLLP 的 AckNak_Seq_Num 字段。之后当发送端收到新的 ACK/NAK DLLP，而且其 AckNak_Seq_Num 字段大于 ACKD_SEQ 计数器的值时（表示发送端至少重传成功一个 TLP），REPLAY_NUM 计数器将被重置为 0，ACKD_SEQ 计数器的值也更新为相应的值。
- PCIe 总线规定发送端新收到的 ACK/NAK DLLP，其 AckNak_Seq_Num 字段不能小于 ACKD_SEQ 计数器，如果出现这种问题，将是芯片的 Bug。
- 如果新的 ACK/NAK DLLP，其 AckNak_Seq_Num 字段值等于 ACKD_SEQ 计数器的值时，表示发送端正在反复地重新发送同一个 TLP，此时 REPLAY_NUM 计数器将加 1，

当这个计数器溢出时，发送端将不再重复发送这个 TLP，而是重新进行链路训练，当 PCIe 链路恢复正常后，再重新发送这个 TLP。

发送端除了设置 REPLAY_NUM 计数器，判断 PCIe 链路可能出现的故障之外，还设置了另一个计数器 REPLAY_TIMER，进一步识别 PCIe 链路可能出现的故障。因为使用 ACKD_SEQ 计数器判断链路故障的基础是发送端可以收到 ACK/NAK DLLP。但是在某种情况下，发送端虽然发送了 TLP，但是接收端没有回应 ACK/NAK DLLP，或者由于 PCIe 链路故障发送端没有收到这个回应。

此时发送端需要使用 REPLAY_TIMER 计数器，以判断在 TLP 的传送过程中是否出现异常。REPLAY_TIMER 计数器记载一个 TLP 报文从发送到获得 ACK/NAK DLLP 回应的时间，当这个时间过长时，发送端认为 PCIe 链路出现故障。REPLAY_TIMER 计数器的更新规则如下所示。

（1）REPLAY_TIMER 计数器的初始值为 0，在发送端发出或者重新发出一个 TLP 后以一个固定的时钟频率开始计数，当 REPLAY_TIMER 计数器到达设定的阈值时，将认为 PCIe 链路出现故障，此时 PCIe 设备将进行 PCIe 链路的恢复工作。如果发送端可以正常收到 ACK/NAK DLLP 时，该计数器不会溢出。

（2）发送端收到 ACK DLLP，而且在 Replay Buffer 中没有“已经发送而且尚未被确认的 TLP”后，REPLAY_TIMER 计数器被重置并重新开始计数。在正常情况下，发送端每发送一个 TLP 后，REPLAY_TIMER 计数器将被重置并重新计数，但是有时在 Replay Buffer 中存在一些 TLP，这些 TLP 已经被发送出去，但是并没有收到相应的 ACK DLLP，此时 REPLAY_TIMER 计数器不能被重置。

（3）当 Replay Buffer 中没有任何 TLP 时，REPLAY_TIMER 计数器将被重置而且其值保持不变。

（4）发送端收到 NAK DLLP 之后，REPLAY_TIMER 计数器将被重置而且其值保持不变。当数据链路层重新发送 TLP 时，REPLAY_TIMER 计数器才重新开始计数。

（5）REPLAY_TIMER 计数器到达设定的阈值后，该计数器的值将被重置，且保持不变。此时 PCIe 链路可能出现某种异常，当这些异常被处理完毕后，REPLAY_TIMER 计数器才可能重新计数。

（6）PCIe 链路重新训练时，REPLAY_TIMER 计数器的值保持不变。准确地说，PCIe 链路处于 Recovery 或者 Configuration 状态时，REPLAY_TIMER 计数器的值保持不变。有关 Recovery 或者 Configuration 状态的详细说明见第 8.2 节。

PCIe 总线规范提供了计算 REPLAY_TIMER 计数器阈值的经验公式，这个阈值和 PCIe 设备和主桥提供的 Max_Payload_Size 成正比，与 PCIe 链路宽度成反比，本节对此公式不进行详细说明。值得注意的是，这个经验公式是基于 x86 处理器计算得出的，不同的处理器在此处的实现不尽相同。

7.2.2 接收端如何使用 ACK/NAK 协议

接收端首先从物理层获得 TLP，此时在这个 TLP 中包含 Sequence 号前缀和 LCRC 后缀。接收端收到这个 TLP 后，首先将这个报文放入 Receive Buffer 中，然后进行 CRC 检查。如果 CRC 检查成功，接收端将根据接收缓冲的阈值发送 ACK DLLP 给发送端，并将这个 TLP 传

给事务层。除了 CRC 校验外，接收端还需要做其他检查，本节对此不进行介绍。

1. 接收端发送 ACK DLLP

当接收端收到的 TLP 没有出现 LCRC 错误，而且 TLP 的 Sequence 号和 NEXT_RCV_SEQ 计数器的值相同时，接收端将正确接收这个 TLP，并将其转发给事务层，随后接收端将 NEXT_RCV_SEQ 计数器加 1。

接收端根据具体情况，决定向发送端立即发送 ACK DLLP，还是等待接收到更多的 TLP 后再发送 ACK DLLP。如果接收端决定发送 ACK DLLP，则该 ACK DLLP 的 AckNak_Seq_Num 字段为 NEXT_RCV_SEQ－1，即已经正确接收 TLP 的 Sequence 号。

接收端不会对每一个正确接收的 TLP 发出 ACK DLLP 回应，因为这样将严重影响 PCIe 总线链路的使用效率，而是收集一定数量的 TLP 后，统一发出一个 ACK DLLP 回应表示之前的 TLP 都已正确接收。

为此接收端使用了一个 ACKNAK_LATENCY_ TIMER 计数器，当这个计数器超时或者接收的报文数超过一个阈值后，向发送端发送一个 ACK DLLP 回应。此时这个 ACK DLLP 的 AckNak_Seq_Num 字段为在这段时间以来，最后一个被正确接收的 TLP 的 Sequence 号。不同的设计在此处的实现不尽相同。但是这些实现都要遵循以下两个原则。

（1）接收端在收到一定数量的报文后，统一发送一个 ACK DLLP 做为回应。

（2）接收端收到的报文虽然没有到达阈值，但是 ACKNAK_LATENCY_ TIMER 计数器超时后，仍然要发出 ACK DLLP 作为回应。在某些情况下，发送端可能在发送一个 TLP 后，在很长一段时间内，都不会发送新的 TLP，此时接收端必须及时给出 ACK DLLP 回应，以免发送端的 REPLAY_TIMER 计数器溢出。

下面将以一个实例说明接收端如何发送 ACK DLLP 回应，该实例如图 7-8 所示，其描述如下所示。

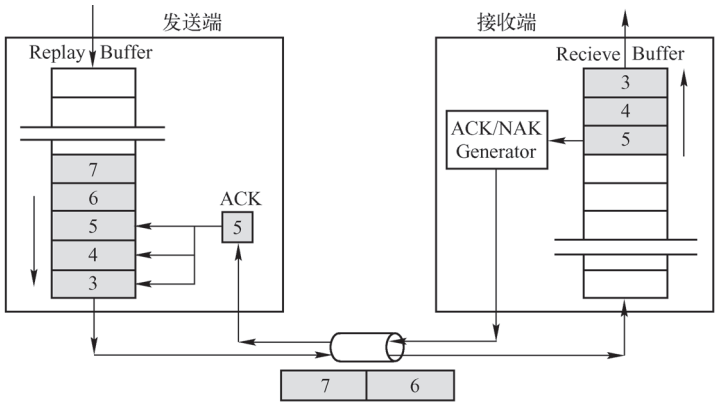

图 7-8　接收端发送 ACK DLLP

（1）发送端发送 TLP3 ~ 7 给接收端，其中 TLP3 是第一个报文，而 TLP7 是最后一个报文。

（2）接收端按序收到 TLP3 ~ 5，而 TLP6 和 7 仍在传送过程中。此时 NEXT_RCV_SEQ 的值被更新为 6，表示下一个即将接收的 TLP，其 Sequence 号为 6。

（3）接收端通过报文检查决定接收 TLP3 ~ 5，然后发送 ACK DLLP，这个 DLLP 的 Ack-

Nak_Seq_Num 字段为 5。为了提高总线的利用率，接收端不会对每一个接收到的 TLP 都做出应答。在这个例子中，AckNak_Seq_Num 字段为 5 表示 TLP3 ~ 5 都已经被接收。

（4）接收端将接收到的 TLP3 ~ 5 传递给事务层。

（5）接收端陆续收到 TLP6 ~ 7 后，继续执行步骤 3 ~ 4。

2. 接收端发送 NAK DLLP

接收端设置了一个 NAK_SCHEDULED 位，该位用来判断接收端如何发送 NAK DLLP，该位的初始值为 0。当接收端接收 TLP 出现错误时，将该位置为 1；当出现“错误的 TLP”被重新接收成功后，该位被置为 0。

如果接收端发现 TLP 的 CRC 错误后，将丢弃这个 TLP，并发送 NAK DLLP，这个 DLLP 的 AckNak_Seq_Num 字段为 NEXT_RCV_SEQ - 1，即最后一个正确接收 TLP 的 Sequence 号。此时 NEXT_RCV_SEQ 计算器将保持不变，NAK_SCHEDULED 位将被置为 1。

当这个 NAK DLLP 到达发送端之前，PCIe 链路还有一些正在传送的 TLP，这些报文将陆续到达接收端，这些报文的 Sequence 号都将大于 NEXT_RCV_SEQ，此时接收端不会接收这些报文，而且接收端也不会为这些报文发送 NAK DLLP，因为此时 NAK_SCHEDULED 位继续保持有效，即为 1。

PCIe 总线规范没有规定接收端如何拒收后续的 TLP 报文，较为合理的实现方式是这些后续的报文无需进入接收端的 Receive Buffer 就被拒绝。这样接收端只为已经进入 Receive Buffer 中的 TLP 发出 ACK/NAK 回应。

如果接收端收到一个重试的 TLP 报文，其 Sequence 号与 NEXT_ RCV_ SEQ 相等，而且报文没有出现错误，接收端将 NEXT_RCV_SEQ 加 1 同时清除 NAK_SCHEDULED 位。此时表示重试的报文已经被正确接收。

如果接收端收到的 TLP，其 Sequence 号小于 NEXT_RCV_SEQ，这种情况通常是由于 TLP 传送过程中的延时，产生的重复 TLP，此时接收端将丢弃这个报文，NEXT_RCV_SEQ 计数器的值也不会改变，随后接收端将向对端发送 ACK DLLP，这个 DLLP 的 AckNak_Seq_Num 为 NEXT_RCV_SEQ - 1。

[Ravi Budruk，Don Anderson and Tom Shanley] 列举了一个实例说明在某种情况下，接收端将收到此类报文。接收端正确接收到 TLP 后，将发送 ACK DLLP，但是由于链路故障，这个报文并没有到达发送端，此时已经发送的 TLP 将不会从发送端的 Replay Buffer 中清除，最终 REPLAY_TIMER 将溢出，此时发送端有可能重新进行链路训练，当链路恢复正常后，发送端将重新发送 Replay Buffer 中的所有 TLP。在这种情况下，接收端将收到 Sequence 号比 NEXT_RCV_SEQ 小的 TLP。

下面将使用一个实例进一步说明接收端如何发送 NAK DLLP，该实例如图 7-9 所示，其描述如下所示。

（1）发送端向接收端发送 TLP3 ~ 7，其中 TLP3 是第一个报文，而 TLP7 是最后一个报文。

（2）接收端按序收到 TLP3 ~ 5，并将这些报文放入 Receive Buffer，当然也可以在这些报文通过完整性检查后，再决定是否将这些 TLP 放入 Receive Buffer 中，而 TLP6 和 7 仍在传送过程中。

（3）接收端通过报文检查决定接收 TLP3 ~ 4，此时 NEXT_RCV_SEQ 为 5，表示即将接收

TLP5。此时接收端将 TLP3 ~ 4 传递给事务层。

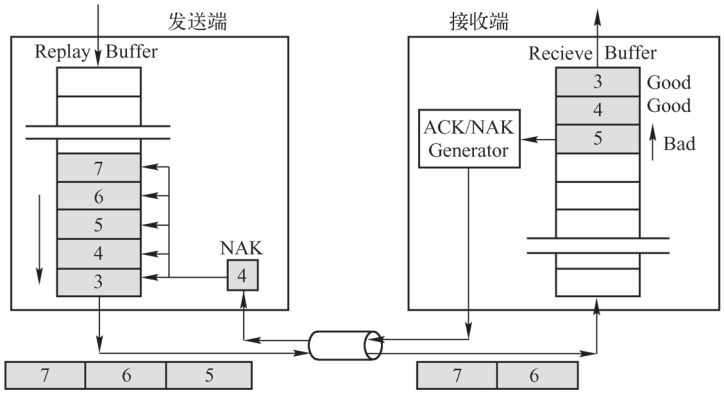

图 7-9　接收端如何发送 NAK DLLP

（4）而 TLP5 没有通过完整性验证，此时接收端将发送 NAK DLLP，这个 DLLP 的 AckNak_Seq_Num 字段为 4，即 NEXT_RCV_SEQ - 1。AckNak_Seq_Num 字段为 4 表示接收端最后一个正确接收的 TLP，其 Sequence 号为 4。此时接收端将设置 NAK_SCHEDULED 位为 1，而 NEXT_RCV_SEQ 保持不变，即为 5。

（5）接收端将丢弃 TLP5。当 TLP6 ~ 7 到达时，接收端仍然丢弃这些报文，即便这些报文通过了完整性检查，因为这些报文的 Sequence 号大于 NEXT_RCV_SEQ。接收端不会为 TLP6 ~ 7 发送 NAK DLLP，因为此时 NAK_SCHEDULED 位有效。

（6）发送端收到 NAK DLLP，其序号为 4，此时发送端首先将 TLP3 ~ 4 从 Replay Buffer 中清除，因为 TLP3 ~ 4 已经被接收端正确接收，然后重新发送 TLP5 ~ 7。

（7）接收端如果正确接收到 TLP5 时，发现其 Sequence 号与 NEXT_RCV_SEQ 相等，将清除 NAK_SCHEDULED 位。

（8）接收端陆续接收到 TLP6 ~ 7，并根据 CRC 的检查结果决定发送 ACK DLLP 或者 NAK DLLP。

在某些情况下，接收端发送的 NAK DLLP 可能并没有被发送端正确接收，因此接收端在很长一段时间内都不会得到“发送端重试的” TLP。此时接收端将会择时重发 NAK DLLP，为此接收端设置了一个 AckNak_LATENCY_TIMER 计数器，当该计数器溢出时，接收端将重发 NAK DLLP。该计数器的更新规则如下。

（1）当接收端发送 ACK 或者 NAK DLLP 时，该计数器重置并开始计数。

（2）接收端为“所有已接收的 TLP” 发送了 ACK DLLP 报文时，或者数据链路层状态为 DL_Inactive 时，该计数器被重置且保持为 0。

AckNak_LATENCY_TIMER 计数器的阈值是 REPLAY_TIMER 计数器阈值的 1/3[⊖]。当接收端等待的时间超过 AckNak_LATENCY_TIMER 计数器的阈值后，接收端将重发一个 ACK

⊖ ACKNAK_LATENCY_TIMER 计数器的阈值是 REPLAY_TIMER 阈值的 1/3，可以保证接收端至少可以重发两次 ACD DLLP 给发送端。

DLLP[⊖]。

当发送端发送若干个 TLP 之后，接收端将发送一个 ACK DLLP 作为回应。但是在某些情况下，发送端并没有收到接收端的 ACK DLLP。此时接收端需要在 AckNak_LATENCY_TIMER 计数器溢出时，重新发送 ACK DLLP。从而防止“因为发送端的 REPLAY_TIMER 计数器溢出”，重新进行 PCIe 链路训练，重发更多的 TLP。

7.2.3 数据链路层发送报文的顺序

数据链路层还规定了报文发送的顺序。由上文的描述中，我们可以发现 DLLP 和 TLP 的发送共用一个 PCIe 链路，除此之外物理层的报文 PLP（Physical Layer Packet）也使用同样的链路。因此 PCIe 链路需要合理地安排报文的发送顺序，以避免死锁。其发送顺序如下所示。

（1）正在发送的 TLP 或者 DLLP 具有最高的优先权。PCIe 总线为了保证数据的完整性，不允许打断正在传送的报文。从理论上讲，打断正在传送的报文是可行的，但是硬件需要更大的代价，也需要制定更加复杂的协议保证数据的完整性。

（2）PLP 的传送。一般来说，处于协议底层的报文优先权高于处于协议高层的报文，这也是解决死锁的一个有效方法。

（3）NAK DLLP。NAK DLLP 需要优先于 TLP 的发送，原理同上。

（4）ACK DLLP。ACK DLLP 响应正确接收的报文，在绝大多数处理过程中，错误处理报文优先于正确的响应，这也是一种防止死锁的方法。

（5）重新传送 Replay Buffer 中的 TLP。也是一种发现错误后的恢复手段，因此这种报文的传递优先权高于其他 TLP。因为在错误没有处理完毕之前，其他 TLP 的传递是没有意义的，接收端都将丢弃这些报文。

（6）其他在事务层等待的 TLP。

（7）其他 DLLP，这些 DLLP 包括地址路由，电源管理等报文，这些报文与数据报文的传递无关，是 PCIe 总线规定的一些控制报文，所以优先权最低。

7.3 物理层简介

如图 4-4 所示，物理层在数据链路层和 PCIe 链路之间，其主要作用有两个，一是发送数据链路层的 TLP 和 DLLP；二是发送和接收在物理层产生的报文 PLP（Physical Layer Packet）；三是从 PCIe 链路接收数据报文并传送到数据链路层。

物理层主要由物理层逻辑模块和物理层电气模块组成，本节主要介绍物理层的逻辑模块，包括 8/10b 编码、链路训练等一些最基础的内容，并通过介绍差分信号的工作原理，简要介绍物理层的电气模块。物理层的电气模块对于深入理解 PCIe 总线规范非常重要，但是许多系统软件工程师因为缺少必要的基础知识，很难理解这部分内容。

本节的内容是第 8 章的基础。如果读者需要深入理解 PCIe 的链路训练，必须掌握本节的全部内容。如果读者对第 8 章内容不感兴趣，可以略过第 8 章和本节。但是 PCIe 总线各个层

⊖ 注意是重发 ACK DLLP 而不是 NAK DLLP。

次间的联系较为紧密，读者很难在对物理层一无所知的情况下，深入理解 PCIe 总线规范。

物理层的电气模块与差分信号的工作原理密切相关，这部分原理包括一系列与信号完整性相关的课题。而信号完整性本身就是一个专门的话题，其难度与复杂程度较高。信号完整性所追求的目标如下。

（1）保证发送的信号可以被接收端正确接收。

（2）保证发送的信号不会影响其他信号。

（3）保证发送的信号不会损坏接收器件。

（4）保证发送的信号不会产生较大的 EMI 电磁噪声。

PCIe 总线的物理层对信号传送进行了一系列约定，以保证信号传递的完整性。而这些约定建立在差分信号传送规则的基础上。如果读者能够深入理解差分信号的工作原理，理解这些约定并不困难。

7.3.1 PCIe 链路的差分信号

PCIe 链路使用差分信号进行数据传递，而差分信号由两个信号组成，这与 PCI 总线使用的单端信号有较大区别。

首先所有信号的传递都需要一个电流回路，而且流入一个节点的电流总和等于流出这个节点的电流总和[㊀]。单端信号使用地平面作为电流回路，而这个地平面并不是稳定的，极易受到干扰，其中最重要的干扰为 SSO（Simultaneous Switching Output）噪声。而减缓 SSO 噪声最有效的方法是为器件的电源提供退耦电容，这个方法也被西方的工程师称为“The rule of thumb”，在电路设计中，该方法极为普及。

即便如此单端信号使用的地平面仍不足以信赖，仍会给单端信号的传递带来不小的干扰。其次单端信号容易收到其他信号的干扰，当单端信号频率较高时，信号在传递过程中衰减较大，而采用差分信号可以有效避免使用单端信号的这些问题。差分信号是由驱动端发送两个等值、相位相反的信号，接收端通过这两个信号的电压差值来判断差分信号是逻辑状态“0”还是“1”。与单端信号相比，差分信号具有许多优势。

（1）抗干扰能力较强。差分信号不受 SSO 噪声的影响，其走线是等长的，且距离较近。当外界存在噪声干扰时，这些干扰同时被耦合到两个信号上，因为接收端只关心这两个信号的差值，这些干扰相减后可以忽略不计。

（2）能够有效抑制信号传递带来的 EMI 干扰。差分信号的极性相反，对外界辐射的电磁场可以相互抵消，因此产生的噪声较小。

（3）逻辑状态定位准确。由于差分信号的开关变化位于两个信号的交点，而不像单端信号使用高低电压两个阈值进行判断，因而受制造工艺、外部环境变化的影响较小，使用差分信号时，接收逻辑较易判断逻辑状态“0”和“1”。

（4）提供的数据带宽较高。由于差分信号受外界环境的影响较小，能够运行在更高的时钟频率上，从而提供的数据带宽较高。

差分信号也有缺点。首先差分信号使用两个信号传递数据，与单端信号相比使用的信号

㊀ 参见基尔霍夫第一定律。

线较多。但是考虑到单端信号为了保证信号质量，往往使用“两线加一地”[㊀]的方式，因此差分信号使用的信号线数量与单端信号相比，并不是简单乘 2 的关系。其次差分信号的布线与单端信号相比具有较多的约束。差分信号对要求等长且平行走线，而且在实际的 PCB 中，最好做到同层等长，因为不同层间的特性阻抗并不完全相等，而是有一定的误差。

这些约束为差分信号的使用带来了一些困难，但是这些困难并不影响差分信号的大规模应用。目前已知的高速链路均使用差分信号，即将问世的 40Gb/100Gb 以太网和 PCIe V3.0 规范也将继续使用差分信号进行数据传递。

差分信号进行传递时依然需要电流回路，而且这个回流路径仍然主要使用地平面，虽然差分信号对彼此也可以作为电流回路，但这并不是主要的回流路径。所有高频信号总是使用电感最小的电流回路，差分信号除了相互间的耦合之外，更多的是对地耦合。

因此差分信号在进行传递时，参考地平面仍然最为重要。如果参考地平面不连续或者不存在参考地平面时，差分信号间的耦合才作为回流通路，但是使用这种方法将极大降低差分信号的质量，而且会增加 EMI 干扰，因此在设计中并不建议使用这种方法。差分信号的传递方法如图 7-10 所示。

如该图所示，差分信号使用两根信号 D+ 和 D- 进行信号传递，其中差分信号可以使用两种方法描述，分别为（V_1，V_2）和（V_{cm}，V_{diff}）。

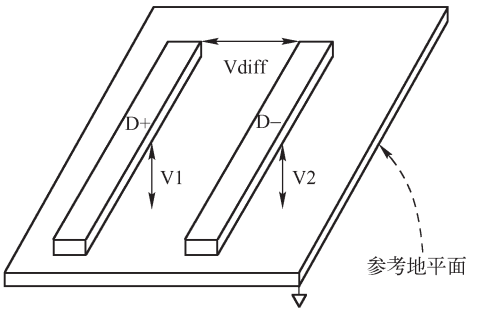

图 7-10　差分信号的传递

假设信号 D+ 的对地参考电压为 V_1，而信号 D- 的对地参考电压为 V_2，使用（V_1，V_2）可以描述一个差分信号，但是这种方法并不常用。因为使用（V_1，V_2）这种方法描述差分信号并不直观，接收端更关心这两个信号的差值，即 $V_1 - V_2$。

为此我们引入两个参数（V_{cm}，V_{diff}），其中 V_{diff}参数代表信号 D + 和信号 D - 的差值电压（Differential Voltage），而 V_{cm}参数代表这两个信号的共模电压（Common Mode Voltage）。这两个参数的计算方法如公式 7-1 所示。

$$\begin{aligned} V_{cm} &= (V1 + V2)/2 \\ V_{diff} &= V1 - V2 \end{aligned} \tag{7-1}$$

其中 $V1 = V_{cm} + V_{diff}/2$ 而 $V2 = V_{cm} + V_{diff}/2$。对于差分信号而言，（V_1，V_2）和（V_{cm}，V_{diff}）这两种描述方式是完全等价的。如图 4-1 所示，在 PCIe 链路中，差分信号首先经过一个 AC 耦合电容后，才能到达接收端。因此接收端收到的差分信号，其直流电压已被滤去。因此在实际应用中，接收端并不关心 V_{cm}，而仅关心 V_{diff}。但是发送端需要置 V_{cm}为一个合适的偏置电压，保证信号的传送。在理想情况下，差分信号 D + 和 D - 相位相反，幅值相等，且 V_{cm}为一个常量，如图 7-11 所示。

在该图中实线部分为 D+ 信号，而虚线部分为 D- 信号，这两个信号相位完全相反，且为非常理想的正弦波，V_{cm}为一个恒定的值，而差分电压 V_{diff}也是一个理想的正弦波，其峰值为 D+ 或者 D- 信号的两倍。

㊀ 使用单端数据总线进行长距离传递时，每两根单端信号线之间使用一根地线隔离。

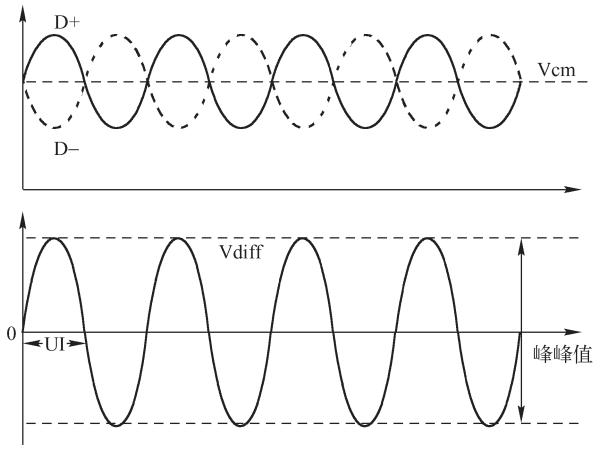

图 7-11　理想的差分信号

然而在差分信号的实际应用中，由于信号 D + 和 D – 并不会完全对称，相位也不会完全相反，可能会存在一些偏差，如图 7-12 所示。

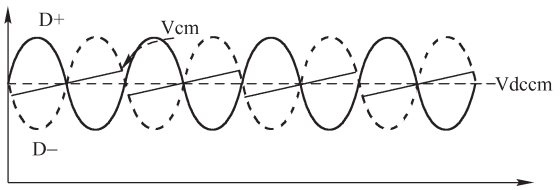

图 7-12　次理想的差分信号

如上图所示，由于 D + 和 D – 并不完全对称，因此 V_{cm}并不是一个常数，而是以 V_{dccm}为中心进行上下波动。V_{dccm}为 DC 共模电压（DC Common Mode Voltage），其值为 V_{cm}在一段时间内的平均直流电压。

与直流共模电压相对应，在对差分信号进行分析时，还使用 AC 共模电压（AC Common Mode Voltage），简写为 Vaccm_rms[1]，其值为 RMS((V1 + V2)-V_{dccm})。其中 RMS（Root Mean Square）用来计算 AC 电压的有效值，对于正弦波而言，RMS 值约等于峰值的 0.707 倍。

在差分信号传递中，使用 UI（Unit Interval）计算单位时间，如图 7-11 所示，UI 的值为一个正弦波的半个周期。在 PCIe V1.x 规范中，使用的时钟频率为 1.25 GHz，因此 UI 在 399.88 ~ 400.12 ps 之间；而 PCIe V2.x 规范使用的时钟频率为 2.5 GHz，此时 UI 在 199.94 ~ 200.06 ps 之间。PCIe 总线还规定了一系列有关差分信号传递的参数，并分为发送逻辑和接收逻辑[2]区别处理，如表 7-3 和表 7-4 所示。

值得注意的是，在本书中发送端与发送逻辑，接收端与接收逻辑是完全不同的概念。如图 4-1 所示，发送端和接收端都包含发送逻辑和接收逻辑，本书为强调发送逻辑和接收逻

[1] 通常交流电压使用有效值表示。

[2] 如图 4-1 所示，发送端和接收端都有相应的发送逻辑（TX）和接收逻辑（RX）。

辑的概念，使用“发送逻辑 TX（Transmitter）和接收逻辑 RX（Receiver）来表示发送端和接收端的发送逻辑和接收逻辑。有的书籍也将发送逻辑和接收逻辑称为发送模块和接收模块。

本节仅列出“发送逻辑 TX”和“接收逻辑 RX”使用的部分参数，对全部参数有兴趣的读者可以参阅 PCIe V2.1 总线规范的表 4-9 和表 4-12。

表 7-3　PCIe 链路“发送逻辑 TX”差分信号的参数

符　号　名	2.5 GT/s	5 GT/s	单　位	描　述
UI	399.88(Min) 400.12(Max)	199.94(Min) 200.06(Max)	ps	时钟误差范围为 ±300 ppm，因此 UI 存在少许误差
$V_{TX-DIFF-PP}$	0.8(Min) 1.2(Max)	0.8(Min) 1.2(Max)	V	V_{diff}的峰峰值，等于 $2\mid V_{D+}-V_{D-}\mid$
$V_{TX-DIFF-PP-LOW}$	0.4(Min) 1.2(Max)	0.4(Min) 1.2(Max)	V	在低电压模式下 V_{diff}的峰峰值
T_{TX-EYE}	0.75(Min)	0.75(Min)	UI	眼图的宽度
$Z_{TX-DIFF-DC}$	80(Min) 120(Max)	120(Max)	Ω	差分信号的 DC 阻抗
$V_{TX-CM-AC-PP}$	Not specified	100(Max)	mV	AC 共模电压的峰峰值，等于 $\max(V_{D+}+V_{D-})/2-\min(V_{D+}+V_{D-})/2$
$V_{TX-CM-AC-P}$	20	Not specified	mV	AC 共模电压的有效值，其值等于 RMS $[(V_{D+}+V_{D-})/2-DC_{AVG}(V_{D+}+V_{D-})/2]$①
$I_{TX-SHORT}$	90(Max)	90(Max)	mA	发送逻辑 TX 在短路状态下的输出电流
$V_{TX-DC-CM}$	0(Min) 3.6(Max)	0(Min) 3.6(Max)	mV	DC 共模电压
$V_{TX-IDLE-DIFF-AC-p}$	0(Min) 20(Max)	0(Min) 20(Max)	mV	发送逻辑 TX 处于 Electrical Idle 状态时，V_{D+}和 V_{D-}的交流电压差值
$V_{TX-IDLE-DIFF-DC}$	Not specified	0(Min) 5(Max)	mV	发送逻辑 TX 处于 Electrical Idle 状态时，V_{D+}和 V_{D-}的直流电压差值。
$V_{TX-RCV-DETECT}$	600(Max)	600(Max)	mV	该参数与 Receiver Detection 的过程相关。第 8.1.3 节将详细介绍 Receiver Detection 逻辑
$T_{TX-IDLE-MIN}$	20(Min)	20(Min)	ns	发送逻辑 TX 处于 Electrical Idle 状态时的最短时间
$T_{TX-IDLE-SET-TO-IDLE}$	8(Max)	8(Max)	ns	发送完毕 EIOS 序列后，发送逻辑 TX 进入 Electrical Idle 状态的最短时间
$T_{TX-IDLE-TO-DIFF-DATA}$	8(Max)	8(Max)	ns	发送逻辑 TX 离开 Electrical Idle 状态，到可以发送正常差分信号需要的转换时间
$T_{CROSSLINK}$	1.0(Max)	1.0(Max)	ms	在使用 Crosslink 连接两个 Switch 时使用
$L_{TX-SKEW}$	500+2UI(Max)	500+4UI(Max)	ps	Lane-Lane 间的传送漂移
C_{TX}	75 (Min) 200(Max)	75(Min) 200(Max)	nF	发送逻辑 TX 使用的 AC 耦合电容

① $DC_{AVG}(V_{D+}+V_{D-})/2$ 为一段时间内 DC 共模电压的平均值，PCIe 总线要求这段时间至少为 10^6 个 UI。

表 7-4　PCIe 链路“接收逻辑 RX”差分信号的参数

符　号　名	2.5 GT/s	5 GT/s	单　　位	描　　述
UI	399.88(Min) 400.12(Max)	199.94(Min) 200.06(Max)	ps	与发送逻辑类似
$V_{RX-DIFF-PP-CC}$	0.175(Min) 1.2(Max)	0.120(Min) 1.2(Max)	V	V_{diff}的峰峰值
$V_{RX-DIFF-PP-DC}$	0.175(Min) 1.2(Max)	0.100(Min) 1.2(Max)	V	
T_{RX-EYE}	0.4(Min)	N/A	UI	眼图的宽度
$T_{RX-MIN-PULSE}$	Not Specified	0.6(Min)	UI	接收端需要的信号最小脉冲间隔
Z_{RX-DC}	40(Min) 60(Max)	40(Min)l60(Max)	Ω	单端信号的 DC 阻抗。该参数的作用是便于发送逻辑 TX 进行 Receiver Detection
$Z_{RX-DIFF-DC}$	80(Min) 120(Max)	Not Specified	Ω	差分信号的 DC 阻抗
$V_{RX-CM-AC-P}$	150(Max)	150(Max)	mV	AC 共模电压
$Z_{RX-HIGH-IMP-DC-POS}$	50k(Min)	50k(Min)	Ω	接收逻辑 RX 的 V_{cc}没有上电时,当输入电压大于 0 时 DC 共模输入阻抗
$Z_{RX-HIGH-IMP-DC-NEG}$	1.0k(Min)	1.0k(Min)	Ω	接收逻辑 RX 的 V_{cc}没有上电时,当输入电压小于 0 时 DC 共模输入阻抗
$V_{RX-IDLE-DET-DIFFp-p}$	65(Min) 175(Max)	65(Min) 175(Max)	mV	用于 Idle 状态检测的电压阈值。其值等于 $2\|V_{RX-D+}-V_{RX-D-}\|$
$T_{RX-IDLE-DET-DIFFENTERTIME}$	10(Max)	10(Max)	ms	当 V_{diff}小于65mV 时，发送逻辑 TX 可能已经处于 Eletrical Idle 状态。发送逻辑 TX 需要在 10 ms 之内识别出这种“意外”的 Electrical Idle 状态①

① 在正常情况下，发送模块进入 Electrical Idle 状态之前,需要发送若干个 EIOS 序列。

以上这些参数将在 PCIe 链路训练和重新训练中使用。在 PCIe 总线中，和差分信号有关的内容还有许多，如阻抗的计算、Emphasis（预加重）、De-Emphasis（预去重）和 PCB 布线等。这些内容并非本书的重点，对此有兴趣的读者可参考 Howard Johnson 和 Martin Graham 合著的 High-Speed Signal Propagation。

深入理解差分信号的工作原理是理解 PCIe 电气子层的重要基础，建议对处理器体系结构有兴趣的读者掌握一些基本的信号完整性的理论知识。从 PCIe 体系结构设计的角度来看，信号完整性这部分内容涉及了许多模拟电路的设计与实现知识，这部分内容是 PCIe 体系结构的精华所在，但并不是本书侧重的内容。

7.3.2　物理层的组成结构

PCIe 总线的物理层通过 LTSSM 状态机对 PCIe 链路进行配置与管理，并与数据链路层进行数据交换，由逻辑子层（Logical Sub-block）和电气子层（Electrical Sub-block）组成。本节主要讲述逻辑子层。逻辑子层与数据链路层进行数据交换，由发送逻辑 TX 和接收逻辑 RX 组成，其结构如图 7-13 所示。

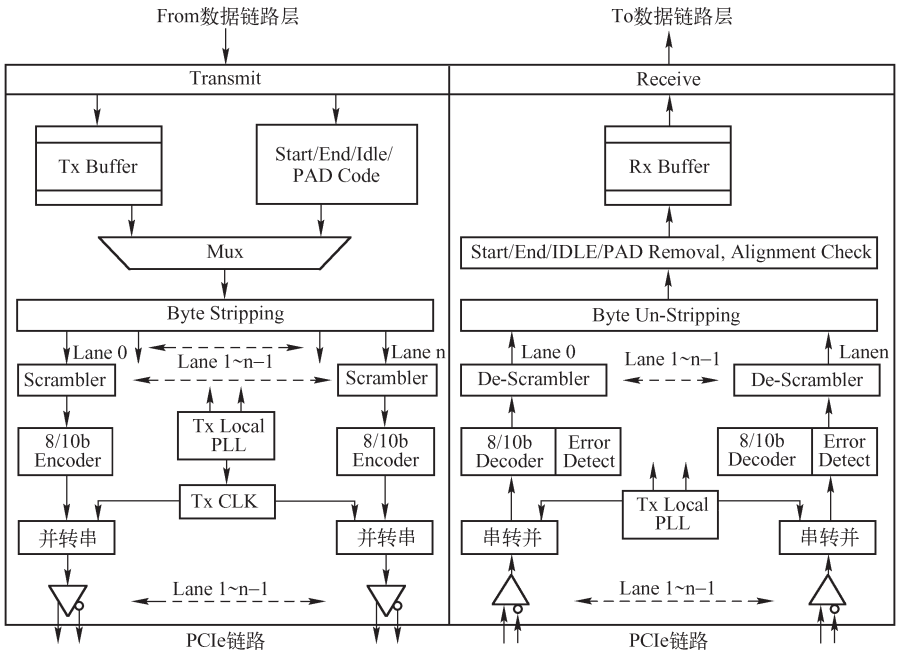

图 7-13　逻辑子层的组成结构

如上图所示，物理层发送报文的过程如下。

（1）物理层从数据链路层获得 TLP 或者 DLLP，然后放入 Tx Buffer 中。

（2）物理层将这些 TLP 或者 DLLP 加入物理层的前缀（Start Code）和后缀（End Code），后通过多路选择器 Mux，进入 Byte Stripping 部件。物理层也定义了一系列 PLP，这些 PLP 也可以通过 Mux，进入 Byte Stripping 部件。

（3）PCIe 链路可能由多个 Lane 组成，Byte Stripping 部件可以将数据报文分发到不同的 Lane 中。在 PCIe 链路的不同 Lane 中传递的数据可能存在漂移，即 Skew，Byte Stripping 部件还有一个重要功能即消除这个漂移，即 De-skew。

（4）数据进入到各自 Lane 的加扰（Scrambler）部件，“加扰”后进行 8/10b 编码，最后通过并转串逻辑将数据发送到 PCIe 链路中。

物理层的接收过程是发送的逆过程，其步骤如下。

（1）物理层从 PCIe 链路的各个 Lane 获得串行数据，并通过 8/10b 解码和 De-Scrambler 部件，发送到“Byte Un-Stripping”部件。

（2）“Byte Un-Stripping”部件将来自不同 Lane 的数据合并，进行 De-skew 操作，然后取出物理层的前后缀并进行边界检查后，将数据放入 Rx Buffer 中。

（3）物理层将在 Rx Buffer 中的数据传递到数据链路层。

物理层的数据在通过 Byte Un-Stripping/Stripping 部件时，需要注意大小端模式的转换。而 Scrambler 和 De-Scrambler 部件的主要作用是对数据流进行“加扰”和“解扰”操作。在串行链路上进行数据传递时，如果在字符流中存在某些规律，这些“规律”将会叠加，并产生较大的 EMI（Electromagnetic interference）噪声。

Scrambler 部件的主要作用就是通过“加扰”的方法削减 EMI 噪声，所谓加扰是指将源

数据流与一个随机序列进行异或操作后，再发送出去。此时被发送出的数据流也基本是伪随机的，从而降低了发送数据时产生的 EMI 噪声。

PCIe 总线通过一个 16 位线性反馈移位寄存器（Linear Feedback Shift Register，LFSR），产生伪随机序列，该移位寄存器的表达式如公式 7-2 所示。

$$G(x) = X^{16} + X^5 + X^4 + X^3 + 1 \tag{7-2}$$

该公式是一个本原多项式，使用该本原多项式可以产生一个周期为 $2^{16} - 1$（这个周期是 16 位移位寄存器能够产生的最大周期）的伪随机序列。所谓本原多项式是“具有最大周期”的不可约多项式。对应的，由本原多项式作为生成多项式所产生的 LFSR 序列为最大周期序列。这些序列一般被称为 m - 序列，在 m - 序列中“0”和“1”所占的比例相对均衡，但是 1 的个数比 0 的个数多 1，因为全 0 不能作为初始值，也不可能是中间状态。

来自 Byte Stripping 部件的字符流与这个伪随机序列中的字符流进行异或操作，从而生成一个相对较为随机的字符流，从而降低了数据流的 EMI 噪声。

De-Scrambler 部件的主要作用是进行解扰。值得注意的是，在 PCIe 链路的两端，加扰和解扰使用的编解码公式相同，而且完全同步，即 LFSR 使用相同的初始值，在 PCIe 链路的两端，该初始值为 0xFFFF。PCIe 链路两端设备每次加解扰一个 8b 数据后，LFSR 进行 8 次移位操作。在 PCIe 总线中，数据在发送时，首先经过“加扰”操作，然后进入 8/10b 编码模块；而接收数据时，首先经过 8/10b 解码模块，然后进行“解扰”操作。

7.3.3 8/10b 编码与解码

IBM 于 1983 年提出 8/10b 编码方法，这个编码方法也是 IBM 的专利。目前这个专利已经过期，以太网、ATM、Infiniband 和 FC（Fiber Channel）在物理链路的数据传送中也使用了 8/10b 编码技术。8/10b 编码是高速串行总线常用的编码方式。

该编码将 8 位编码转化为 10 位，以平衡数据流中 0 与 1 的数量。使用这种方法可以保证数据流中 1 和 0 的数量相等，即保证直流平衡（DC Balance）。如果在一个高速串行数据流中有较多连续的“1”时，会将 AC 耦合电容充满，从而影响这些电容的正常工作，在 PCIe 链路上，AC 耦合电容的位置如图 4-1 所示。

PCIe V1. x 和 2. x 规范使用了 8/10b 编码方式，而 V3. 0 规范将使用 128/130b 编码方式。128/130b 编码方式与 8/10b 编码方式原理较为类似，使用 128/130b 编码可以进一步提高 PCIe 总线的利用率，但是需要更多的硬件资源。本节仅介绍 8/10b 编解码方式。在 PCIe 总线中，编码与解码的过程如图 7-14 所示。

8/10b 编码的基本原理是将一个连续的 8 位数据流分为两组，其中一组由 3 位（FGH）组成，而另一组由 5 位（ABCDE）组成。8/10b 编码将这两组数据流分别编码成一组 4 位（fghj）和一组 6 位（abcdei）的数据流。而解码过程是编码的逆过程，将一组 4 位和一组 6 位的数据流还原成为一组 3 位和一组 5 位的数据流。

PCIe 设备采用这种编码方式可以保证数据流中出现的 0 和 1 的数量基本保持一致，同时保证在通过 PCIe 物理链路的数据流中，连续的“1”和“0”不会超过 5 个。虽然采用 8/10b 编码将降低总线的使用效率，但是能够保证高速串行信号的传送完整性。

如图 7-13 所示，物理层可以对两类字符进行 8/10b 编码，一类是数据字符，即从数据链路层获得的 TLP 和 DLLP；一类是物理层使用的控制字符，如 Start/End/Idle Code

和一些物理层中使用的 PLP。为此 PCIe 总线在进行 8/10b 编解码需要区分数据和控制字符。

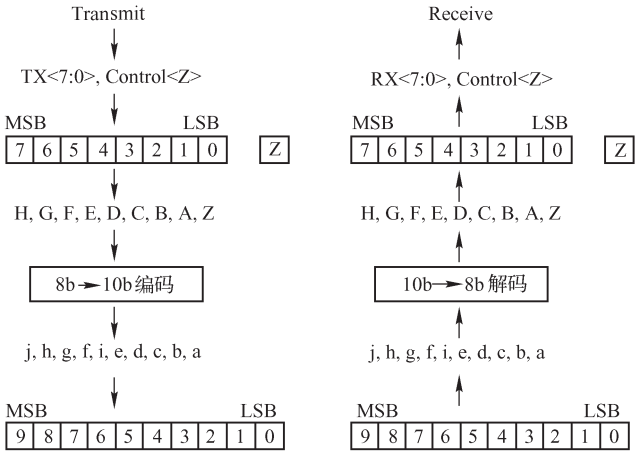

图 7-14　8/10b 编解码过程

数据字符与控制字符使用的 8/10b 编码不同，PCIe 总线分别使用 Dxx. y 和 Kxx. y 表示数据字符（D）和控制字符（K），其中 xx 记录字符的低 5 位 ABCDE，而 y 记录字符的高 3 位 FGH。

值得注意的是，PCIe 总线使用 8/10b 编码可以保证每十位中，最多有 6 个 1 或者 6 个 0，而不是传统 8/10b 编码中要求的“不超过 5 个连续的 0 或者 1”。使用这种方法基本上可以保证数据流的 DC 平衡。但是使用这种编码方式无法保证在某些特殊情况中，连续发送的数据流中都含有“6 个 1，4 个 0”或者“6 个 0，4 个 1”。随着时间的累积，这些数据流依然会在链路中造成严重的 DC 失衡。

为此 PCIe 总线使用 CRD（Current Running Disparity）技术进一步保证 PCIe 链路的 DC 平衡。PCIe 总线在进行 8/10b 编码时，每一个 Dxx. y 和 Kxx. y 对应两个 10b 的编码，分别是 CRD + 和 CRD – 。对于多数编码，CRD – 和 CRD + 中含有的“0”和“1”的个数相同，如 D1. 0、D2. 0 等数据字符。但是在有些 CRD + 编码中，“1”的个数小于“0”的个数；而在 CRD – 编码中“0”的个数大于“1”的个数，如 D1. 1 和 D2. 1 的编码。

在 CRD + 编码中，“1”的个数小于或者等于“0”的个数；在 CRD – 编码中，“0”的个数小于或者等于“1”的个数。值得注意的是，CRD + 和 CRD – 编码并不是直接取反的关系，当 CRD 编码的“0”的个数与“1”的个数相同时，CRD + 与 CRD – 的编码有时是相同的，如 D3. 5 的编码。

下文以一个数据发送的实例说明 CRD + 、CRD – 编码的使用。在 PCIe 链路的发送端中，存在一个 CRD 状态位，其初始值可以为“正”或者“负”。随着通过 PCIe 链路的数据流增多，累积的“1”和“0”的个数可能并不平衡。当所有通过 PCIe 链路的字符流中，“1”的个数大于“0”的个数时，CRD 状态为正；当所有通过 PCIe 链路的字符流中，“0”的个数大于“1”的个数时，CRD 状态为负；当所有通过 PCIe 链路的字符“0”的个数等于“1”的个数时，CRD 状态保持不变。

当 CRD 状态为正时，PCIe 链路进行 8/10b 编码时使用 CRD +，否则使用 CRD – 以维持 PCIe 链路的 DC 均衡。在 PCIe 总线中，数据字符使用的 8/10b 编码的格式如表 7–5所示。

表 7–5　数据字符的 8/10b 编码

数 据 字 符	Data Byte	HGF EDCBA	CRD – abcdei fghi	CRD + abcdei fghi
D0. 0	0x00	000 00000	100111 0100	011000 1011
D1. 0	0x01	000 00001	011101 0100	100010 1011
D2. 0	0x02	000 00010	101101 0100	010010 1011
D3. 0	0x03	000 00011	110001 1011	110001 0100
D4. 0	0x04	000 00100	110101 0100	001010 1011
D5. 0	0x05	000 00101	101001 1011	101001 0100
D6. 0	0x06	000 00110	011001 1011	011001 0100
D7. 0	0x07	000 00111	111000 1011	000111 0100
…				
D1. 1	0x21	001 00001	011101 1001	100010 1001
D2. 1	0x22	001 00010	101101 1001	010010 1001
D3. 1	0x23	001 00011	110001 1001	110001 1001
…				
D3. 5	0xA3	101 00011	110001 1010	110001 1010
…				
D23. 7	0xF7	111 10111	111010 0001	000101 1110
D24. 7	0xF8	111 11000	110011 0001	001100 1110
D25. 7	0xF9	111 11001	100110 1110	100110 0001
D26. 7	0xFA	111 11010	010110 1110	010110 0001
D27. 7	0xFB	111 11011	110110 0001	001001 1110
D28. 7	0xFC	111 11100	001110 1110	001110 0001
D29. 7	0xFD	111 11101	101110 0001	010001 1110
D30. 7	0xFE	111 11110	011110 0001	100001 1110
D31. 7	0xFF	111 11111	101011 0001	010100 1110

PCIe 总线还定义了一系列控制字符，这些字符从“Data Byte”的角度来看和数据字符完全相同，但是使用的 CRD + 和 CRD – 编码和数据字符不同。如数据字符 D30. 7 和 K30. 7 所对应的“Data Byte”都为 0xFE（111 11110），但是 CRD – 编码分别为 011110 0001 和 011110 1000，而 CRD + 编码为 100001 1110 和 100001 0111。

控制字符使用的 8/10b 编码的格式如表 7–6 所示。PCIe 总线使用这些字符编码作为控制命令，和数据进行区别。

表 7-6　控制字符的 8/10b 编码

数据字符	Data Byte	HGF EDCBA	CRD - abcdei fghi	CRD + abcdei fghi
K28. 0	0x1C	000 11100	001111 0100	110000 1011
K28. 1	0x3C	001 11100	001111 1001	110000 0110
K28. 2	0x5C	010 11100	001111 0101	110000 1010
K28. 3	0x7C	011 11100	001111 0011	110000 1100
K28. 4	0x9C	100 11100	001111 0010	110000 1101
K28. 5	0xBC	101 11100	001111 1010	110000 0101
K28. 6	0xDC	110 11100	001111 0110	110000 1001
K28. 7	0xFC	111 11100	001111 1000	110000 0111
K23. 7	0xF7	111 10111	111010 1000	000101 0111
K27. 7	0xFB	111 11011	110110 1000	001001 0111
K29. 7	0xFD	111 11101	101110 1000	010001 0111
K30. 7	0xFE	111 11110	011110 1000	100001 0111

这些控制字符在 PCIe 总线中的定义如表 7-7 所示。

表 7-7　控制字符的说明

数 据 字 符	缩　　写	符　号　名	说　　明
K28. 0	SKP	Skip	用于补偿 PCIe 链路不同 Lane 的延时，PCIe 总线的物理层收到该控制序列时，LFSR 不进行移位操作
K28. 1	FTS	FTS（Fast Training Sequence）	在链路训练的 FTS 序列中使用
K28. 2	SDP	Start DLLP	DLLP 的起始标记
K28. 3	IDL	Idle	在 EIOS（Electrical Idle OrderedSet）序列中使用
K28. 4			保留
K28. 5	COM	Comma	复位 PCIe 链路的 LFSR 为初始值
K28. 6			保留
K28. 7	EIE	Electrical Idle Exit	在 EIEOS（Electrical Idle Exit Sequence）序列中使用
K23. 7	PAD	Pad	填充字符
K27. 7	STP	Start TLP	TLP 的起始标志
K29. 7	END	End	TLP 和 DLLP 的结束标志
K30. 7	EDB	EnD Bad	无效 TLP 的结束标志

下文以物理层发送一个 TLP 的实例说明，PCIe 链路如何使用这些编码。一个 TLP 在通过物理层时，首先要加入物理层的前后缀，分别为 STP 和 END。加入这些前后缀后的 TLP 报文格式如图 7-15 所示。

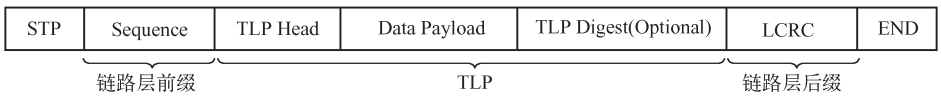

图 7-15　物理层 TLP 的格式

TLP 在通过物理层时首先在其前后加入 STP 和 END 控制字符，这两个控制字符分别为 K27.7 和 K29.7（如表 7-7 所示），它们通过物理层时，不需要进行加接扰操作。数据链路层前缀、TLP 和数据链路层后缀都属于数据字符，这些字符在通过物理层时需要进行加接扰操作，之后从表 7-5 中获得字符流，并由物理层发向 PCIe 链路。

值得注意的是，控制字符和数据字符需要根据物理层 CRD 状态，决定是使用 CRD + 还是 CRD - 编码。PCIe 链路的两端在进行加解扰操作时，需要保证其使用的 LFSR 寄存器同步。LFSR 寄存器的同步由控制字符 COM 决定，在初始复位时 LFSR 寄存器的初始值为 0xFFFF，当收到控制字符 COM 后，物理层将 LFSR 寄存器的初始值置为 0xFFFF，此外物理层收到控制字符 SKP 后，并不会对 LFSR 寄存器进行移位操作。

7.4 小结

本章重点介绍了数据链路层的状态，以及 ACK/NAK 协议，并简要介绍了 PCIe 总线的物理层。其中 PCIe 总线的物理层非常重要，深入理解物理层是深入理解 PCIe 体系结构的要点。在第 8 章讲述的内容以此为基础。

第 8 章　PCIe 总线的链路训练与电源管理

PCIe 链路的初始化过程较为复杂。PCIe 总线进行链路训练时将初始化 PCIe 设备的物理层、发送接收模块和相关的链路状态信息，当链路训练成功结束后，PCIe 链路两端的设备可以进行正常的数据交换。

链路训练的过程由硬件逻辑完成，而无需系统软件的参与。此外当 PCIe 设备从低功耗状态返回到正常工作模式时，或者 PCIe 链路出现某些错误时，PCIe 链路也需要重新进行链路训练。

8.1　PCIe 链路训练简介

PCIe 总线进行链路训练的主要目的是初始化 PCIe 链路的物理层、端口配置信息、相应的链路状态，并了解链路对端的拓扑结构，以便 PCIe 链路两端的设备进行数据通信。一条 PCIe 总线提供的链路带宽可以是 ×1、×2、×4、×8、×12 或者 ×16，但是在这个 PCIe 链路上所挂接的 PCIe 设备并不会完全使用这些链路。如一个 ×4 的 PCIe 设备可能会连接到 ×16 的 PCIe 链路上。此时该 PCIe 设备在进行链路训练时，必须通知对端链路该设备实际使用的链路状态。

此外 PCIe 总线规定，PCIe 链路两端的设备所使用的 Lane 可以错序进行连接，PCIe 总线规范将该功能称为“Lane Reversal”。在相同的 Lane 上，差分信号的极型也可以错序连接，PCIe 总线规范将该功能称为 Polarity Inversion。这两种错序连接方式如图 8-1 所示。

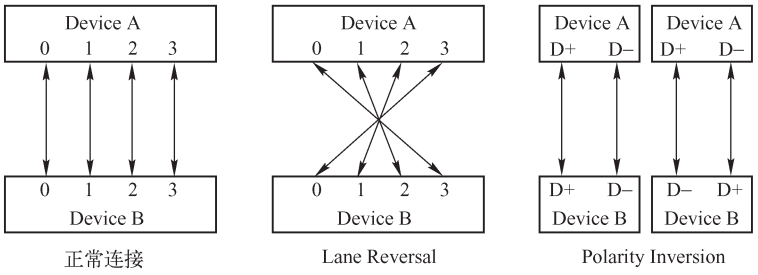

图 8-1　PCIe 设备的错序连接

PCIe 总线提供这些连接方式的主要目的是为了方便 PCB 走线，因为差分信号要求在 PCB 中等长而且等距。在一个系统中，如果存在多路差分信号时，PCB 布线较为困难。PCIe 链路允许“Lane Reversal”和“Polarity Inversion”这两个功能，便于 PCB Layout 工程师根据实际情况为差分信号选择更为合理的走线路径，从而降低 PCB 的层数。除了 PCIe 链路，还有许多使用差分信号的串行总线也支持“Lane Reversal”和“Polarity Inversion”这两个功能，但是称呼上有所区别。在一条 PCIe 链路中，可以同时支持“Lane Reversal”和“Polarity Inversion”这两个功能。

PCIe 链路进行链路训练时，需要了解 PCIe 链路两端的连接拓扑结构。一条 PCI 链路可能使用多个 Lane 进行数据交换，而数据报文经过不同 Lane 的延时并不完全相同。PCIe 总线进行链路训练时，需要处理这些不同 Lane 的延时差异，并进行补偿。PCIe 总线规范将这个过程称为 De-skew。

此外 PCIe 总线在链路训练过程中，还需要确定数据传送率。PCIe V1. x 总线使用的数据传送率为 2. 5GT/s，PCIe V2. 0 总线使用 5. 0GT/s，而 PCIe V3. 0 总线使用 8GT/s 的数据传送率。当分属不同规范的 PCIe 设备使用同一个 PCIe 链路进行连接时，需要统一数据传送率。如一个 V1. x 的 PCIe 设备与一个 V2. 0 的 RC 或者 Switch 连接时，需要将数据传送率统一为 2. 5GT/s。在 PCIe 总线中，如果一个 PCIe 链路的两端分别连接不同类型的 PCIe 设备时，将选择较低的数据传送率。值得注意的是，PCIe 链路在进行初始化时，首先使用 2. 5GT/s 的数据传送率，之后切换到更高的数据传送率，如 5GT/s 或者 8GT/s。

在讲述 PCIe 链路训练之前，读者需要了解一些与 Link Number 和 Lane Number 相关的基本概念。在多端口 RC 和 Switch 中具有多个下游端口，而每个端口可以支持 ×1、×2、×4 等不同宽度的 Lane，如图 8-2 所示。

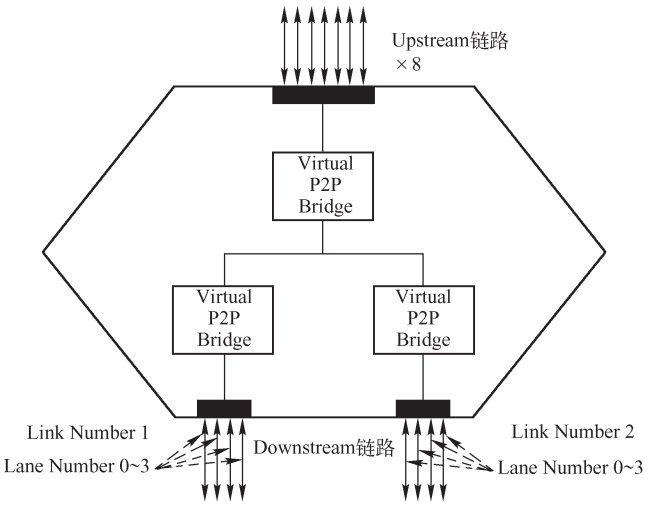

图 8-2　Link Number 和 Lane Number

在一个 Switch 中存在多个下游链路，并使用 0 ~ n 进行编号，其中 n≤255。这些编号保存在 Switch 的硬件逻辑中，而不在 Switch 的配置空间中。这个编号也被称为 Link Number，上图所示的 Switch 中含有两个 Link Number，分别为 1 和 2。

在 Switch 中，还有两类 Lane Number，分别是物理“Lane Number”和逻辑“Lane Number”。其中物理“Lane Number”是链路训练之前使用的 Lane number。一个 PCIe 链路的物理“Lane Number”编号为 0 ~ n，其中 n 为 PCIe 链路的最大 Lane Number，如果一个 PCIe 链路上有 8 个 Lane，则 n 等于 7。

而逻辑“Lane Number”是链路训练结束后使用的 Lane Number。如图 8-1 左图所示，PCIe 链路允许错序连接，因此物理“Lane Number”与逻辑“Lane Number”并不相同。物

理“Lane Number”与逻辑“Lane Number”的对应关系在链路训练中确定。

除此之外，有些 Switch 支持多种链路配置方式，假设某个 Switch 的下游支持 8 个 Lane。这 8 个 Lane 可以组成 1 个 PCIe 链路，其链路宽度为 8；也可以组成 2 个 PCIe 链路，每个链路宽度为 4；也可以组成 4 个链路，每个链路宽度为 2。

该 Switch 的物理 Lane Number 的编号方法不变，都是从 0 ~ 7，但是逻辑 Lane Number 的编号方法将有所区别。如图 8-2 所示的 Switch 中具有 8 个 Lane，组成两个 PCIe 链路，其中每条 PCIe 链路的逻辑 Lane Number 的编号都为 0 ~ 3。

PCIe 总线进行链路训练时，需要进行 RC 或者 Switch 的 Link Number 和 Lane Number 的初始化，在第 8.2 节中将详细介绍这些内容。

PCIe 总线进行数据传递时，需要使用时钟进行同步，但是 PCIe 链路并没有提供这个时钟信号，因此在进行链路训练时，接收端需要从发送端的数据报文中提取接收时钟。PCIe 总线规范将这个获得接收时钟的过程称为“Bit Lock”。

在链路训练过程中，PCIe 链路需要首先确定 COM 字符，该字符也标志着链路训练或者链路重训练的开始，PCIe 总线规范将确定 COM 字符的过程称为“Symbol Lock”。如表 7-6 所示，COM 字符为“001111 1010”或者“110000 0101”，该字符为 2 个“0”后 5 个“1”或者 2 个“1”后 5 个“0”，非常便于硬件识别。Bit Lock 和 Symbol Lock 的过程也需要在 PCIe 总线的链路训练中进行。

8.1.1 链路训练使用的字符序列

PCIe 总线进行链路训练时，需要发送一些特殊的字符序列（Ordered-Sets），这些 Oder-er-Sets 将在下文中详细介绍，PCIe 总线规范定义了以下几类 Ordered-Sets。有的书籍也将这些 Ordered-Sets 称为 PLP，即物理层报文。

- Training Sequence 1 和 2，简称为 TS1 和 TS2 序列。这两种 PLP 在链路训练的多个状态机中使用，下文将进一步介绍这两种字符序列。
- Idle 序列。在正常情况下，发送端进入 Electrical Idle 状态时，将首先向对端发送若干 Idle 序列，才能进入。Electrical Idle 状态是 PCIe 链路的一个低功耗状态，第 8.1.2 节将详细介绍该状态。
- Fast Training Sequence，简称为 FTS。该字符序列协助接收逻辑获得 Bit/Symbol Lock，接收逻辑需要获得多个 FTS 后，才能确定 Bit/Symbol Lock。
- SKIP 序列。该字符序列的主要作用是进行时钟补偿。

在 PCIe 总线中，字符序列的发送方式与 TLP 和 DLLP 有较大不同。假设一条 PCIe 链路由多个 Lane 组成，那么 TLP 和 DLLP 报文将分散到多个 Lane 中。而字符序列必须同时出现在这些不同的 Lane，这几个 Lane 必须“在同一个时间点”发送字符序列。而不能出现一个 Lane 正在发送这个字符序列进行与链路训练相关的操作，而其他 Lane 进行其他数据传递的情况。PCIe 链路发送 TLP 与发送字符序列的过程如图 8-3 所示。

如在一个 ×4 的 PCIe 链路中发送 SKIP 序列时，每一个 Lane 中都要出现“COM、SKP、SKP、SKP”这样的数据流。其他字符序列的发送方法也与此类似。而 TLP 或者 DLLP 的发送分散到各个 Lane 上。

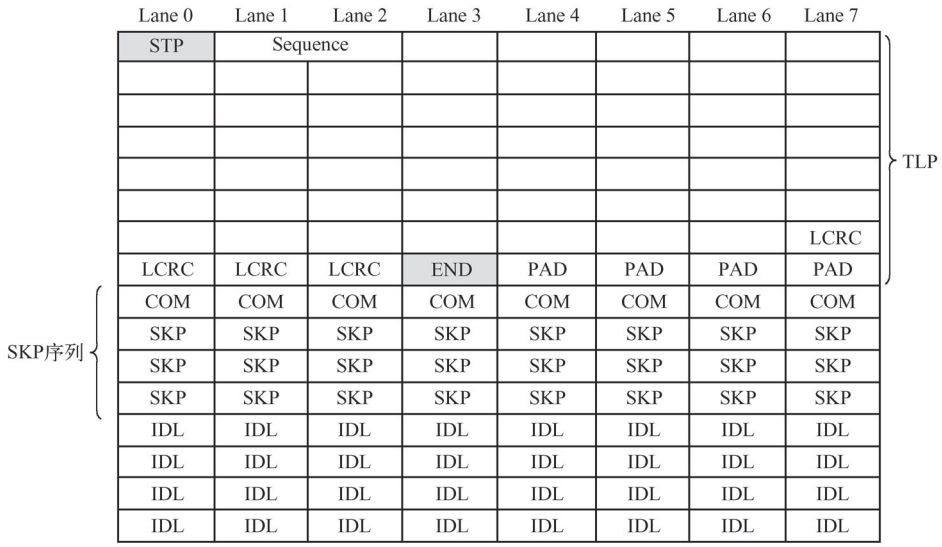

图 8-3　特殊字符序列的发送

1. TS1 和 TS2 序列

在物理层的 LTSSM 状态机中，TS1 和 TS2 序列的使用方法不同。其中 TS1 序列的主要作用是检测 PCIe 链路的配置信息，而 TS2 序列确认 TS1 序列的检测结果。

TS1 和 TS2 序列由 16 个字符组成，单纯从结构上看，TS1 和 TS2 仅仅是第 6 ~ 15 个字符的含义不同，但是这两个序列在 LTSSM 状态机中的使用方法不同。TS1 的第 6 ~ 15 个字符为 D10. 2，而 TS2 的第 6 ~ 15 个字符为 D5. 2，其中 D10. 2 和 D5. 2 也是 TS1 和 TS2 的标识号。TS1 和 TS2 序列的其他字符如下所示。

- 第 0 字符为 COM 控制字符，表示 TS1/TS2 序列的开始。TS1/TS2 字符序列将复位 LFSR 寄存器。
- 在链路训练的初始阶段，第 1 字符存放控制字符 PAD，即为空。而在链路的配置阶段，该字符存放端口使用的 Link Number。
- 在链路训练的初始阶段，第 2 字符为控制字符 PAD，即为空。而在链路的配置阶段，该字符存放端口使用的 Lane Number。
- 第 3 个字符为 FTS 序列的个数（N_FTS）。不同的 PCIe 链路需要使用不同数目 FTS 序列，才能使接收端的 PLL 锁定接收时钟。
- 第 4 个字符存放当前 PCIe 设备支持的数据传送率，第 1 位为 1 表示支持 2. 5GT/s 传送率；第 2 位为 1 表示支持 5GT/s 传送率；第 3 位为 1 表示支持 8GT/s 的数据传送率（在 PCIe V3. 0 规范中使用）；第 4 ~ 5 位保留；第 6 位是一个多功能位，当 PCIe 链路的没有配置成功时可以作为 Notification 位，也可以用作发送链路 De-emphasis 的使能位；第 7 位为 speed_change 位，当该位为 1 时，通知 PCIe 链路对端设备需要改变传送速率。
- 第 5 个字符存放命令。第 0 位为“Hot Reset”，第 1 位为“Disable Link”，第 2 位为“Loopback”，第 3 位为“Disable Scrambling”，第 4 位为“Compliance Receive”。当接收逻辑 RX 收到 TS1 或者 TS2 序列后，将根据该字符的命令进行对应的操作。

2. Idle 序列

在正常情况下，当发送端进入 Electrical Idle 状态之前，必须向对端发送 EIOS 序列。如果 PCIe 设备使用 2.5GT/s 的传送率时，Idle 序列由 1 个 COM 字符加 3 个 IDL 字符组成，即“COM IDL IDL IDL”；如果 PCIe 设备使用 5GT/s 的传送率时，Idle 序列由两组这样的字符序列组成，即“COM IDL IDL IDL COM IDL IDL IDL”。Electrical Idle 状态是一种特殊的 Idle 状态，处于该状态时，PCIe 链路使用的功耗最低，该状态的详细解释见第 8.1.2 节。

当发送端退出 IDLE 状态时，必须向对端发送 EIEOS 序列。EIEOS 序列仅在链路传送率大于 2.5GT 时使用，该序列由 1 个 COM 字符、14 个 EIE 字符和 D10.2（TS1 识别符）组成。

PCIe 设备可以根据链路的使用情况确定当前链路是否处于 Electrical Idle 状态，而不是必须收到 Idle 序列后进入该状态。如一个 PCIe 设备在很长一段时间没有收到流量控制报文或者链路处于 Electrical Idle 状态时，也可以推断出对端设备处于 Idle 状态。

3. FTS 序列

单个 FTS 序列由 1 个 COM 字符加 3 个 FTS 字符组成，该序列的主要目的是使接收逻辑 RX 重新获得 Bit/Symbol Lock。发送逻辑需要向对端发送多少个 FTS 序列由接收到的 TS1/2 序列决定，TS1/2 序列的第 3 个字符为需要发送 FTS 序列的个数。

4. SKIP 序列

SKIP 序列由一个 COM 字符加 3 个 SKP 字符组成。物理层提供 SKIP 序列的主要原因是进行时钟补偿。假设一个 PCIe 设备使用的时钟频率为 2.5GHz ± 300ppm，其中 300ppm 意味着这个时钟源每发出 1 百万个时钟可能产生 300 个时钟漂移，即每 3333 个时钟将可能产生一个时钟漂移。如果 PCIe 链路不使用 SKIP 序列，本地时钟与“从报文中提取”的时钟存在的漂移，可能导致数据传送失败。

在 PCIe 设备的接收逻辑 RX 中，使用了两个时钟，一个时钟是通过 PLL 从接收报文中恢复的时钟，另一个时钟是接收逻辑 RX 使用的本地时钟。这两个时钟间的关系如图 8-4 所示。值得注意的是这两个时钟并不完全同步。

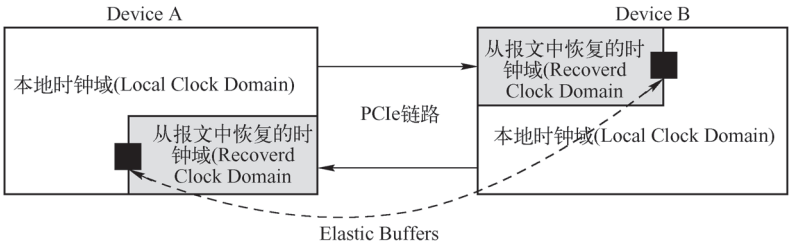

图 8-4　本地时钟域与从报文中恢复的时钟域

在 PCIe 总线中，使用 Elastic Buffer[⊖]技术处理这两个时钟之间的频率差和相位差。Elastic Buffer 处于本地时钟域与“被恢复的”时钟域之间，由一个同步 FIFO 组成。该 FIFO 的一端使用本地时钟域、而另一端使用“被恢复的”时钟域。其中本地时钟与“被恢复的”时钟频率都为 2.5GHz ± 300 ppm。

但是如果 PCIe 设备从数据报文中恢复的时钟频率为 2.5GHz - 300ppm，而本地时钟频率

⊖ Elastic Buffer 也被称为 Elasticity Buffer 或者 Synchronization Buffer。

为 2.5 GHz + 300 ppm 时，Elastic Buffer 两端的时钟频率并不匹配，Elastic Buffer 将出现 Overrun 的现象；而如果本地时钟频率小于“被恢复的”时钟时，Elastic Buffer 将可能出现 Underrun 的现象。如果 PCIe 总线不采取一些必要的补救措施，那么无论 Elastic Buffer 的容量有多么大，都可能出现 Overrun 和 Underrun 的现象。

为此，PCIe 总线规定，物理层的每个 Lane 发送 1180 ~ 1538 个字符之后，必须发送一个 SKIP 序列进行时钟补偿。因为在最恶劣的情况下，接收逻辑 RX 每过 1667 个时钟周期，本地时钟就可能与“被恢复的”时钟相差一个时钟周期[㊀]。在 PCIe 总线中，当 Elastic Buffer 收到 SKIP 序列时，可以根据自身的状态选择是增加还是减少 1 ~ 2 个 SKIP 序列，从而补偿本地时钟与“被恢复的”时钟之间的频率与相位差。

在一个具体的实现中，可以通过计算，得到 Elastic Buffer 不出现 Overrun 和 Underrun 所需要的最小尺寸。这个最小尺寸与 SKP 序列的发送间隔（多少个时钟周期发送一次 SKP 序列），Max_Payload_Size 参数和 PCIe 链路的数据传送率相关。对此有兴趣的读者可以参考 Elastic Buffer Implementations in PCI Express Devices，以获得详细的量化分析结果，本节对此不做进一步说明。

Elastic Buffer 技术由来已久，除了 PCIe 总线之外，USB 总线、InfiniBand、Fibre Channel 和 Gigabit Ethernet 中也使用该技术处理分属不同时钟域的数据传递。

8.1.2 Electrical Idle 状态

当发送端或者接收端进入 Electrical Idle 状态后，两端的发送逻辑 TX 将驱动 D+ 和 D- 信号的对地电压为相同的值，其值等于 DC 共模电压（DC 共模电压的定义见第 7.3.1 节），从而使发送逻辑 TX 处于“最低功耗”状态。

当发送端处于正常工作模式时，D+ 和 D- 信号差值电压的峰峰值为 $V_{TX\text{-}DIFF\text{-}PP}$，其值在 0.8 V ~ 1.2 V 之间，而当发送端处于 Electrical Idle 状态时，D + 和 D - 信号差值电压的峰峰值为 $V_{TX\text{-}IDLE\text{-}DIFF\text{-}DC}$，其值在 0 ~ 5 mv 之间。

由此可以发现当发送端处于 Electrical Idle 状态时，在示波器中显示的 D + 和 D - 信号的对地电压基本相等，此时发送端处于完全“静止”状态，在发送逻辑 TX 中基本没有电流通过，因此从理论上讲处于“最低功耗”状态。

在正常情况下，当发送端或者接收端进入 Electrical Idle 状态之前，都需要使用发送逻辑 TX 向对端发送若干 EIOS 序列，之后才能进入该状态。当发送逻辑 TX 处于 Electrical Idle 状态时，可以处于高阻抗或者低阻抗模式，PCIe 总线规范对此并没有限制。而接收逻辑 RX 处于 Electrical Idle 状态时，其 DC 共模输入阻抗必须在 PCIe 总线规范要求的范围内。

当发送逻辑 TX 进入 Electrical Idle 状态后，至少需要经过 TTX-IDLE-MIN 这段时间延时后，才允许退出该状态，因为对端接收逻辑 RX 的“Electrical Idle Exit detector”部件从启动到正常工作的时间延时为 TTX-IDLE-MIN。如果在“Electrical Idle Exit detector”部件没有正常工作之前，发送逻辑 TX 就退出 Electrical Idle 状态，对端的接收逻辑 RX 将不能检查到这个状态变化，从而导致错误。

㊀ 因为本地时钟与“被恢复的”时钟间可能相差 600 ppm。

发送逻辑 TX 在退出 Electrical Idle 状态时，必须在 $T_{TX\text{-}IDLE\text{-}TO\text{-}DIFF\text{-}DATA}$时间范围内，需要为 D + 和 D – 信号提供正常的工作伏值，D + 和 D – 信号的峰峰值至少为 $V_{TX\text{-}DIFF\text{-}PP}$，其值在 0.8 V ~1.2 V 之间。而对端的接收逻辑 RX 发现其 $V_{RX-IDLE\text{-}DET\text{-}DIFFp\text{-}p}$的值小于 75 mV 时，将不会退出 Electrical Idle 状态，只有当 $V_{RX\text{-}IDLE\text{-}DET\text{-}DIFFp\text{-}p}$的值大于 175 mV 时，对端的接收逻辑 RX 才会退出 Electrical Idle 状态。

值得注意的是，发送逻辑 TX 经过发送链路将信号传递到接收逻辑 RX 时，信号将会衰减。虽然发送逻辑 TX 输出的 VTX-DIFF-PP 在 0.8 V ~1.2 V 之间，但是接收端收到的 VRX-IDLE-DET-DIFFp-p，其最小值可能只有 175 mV 左右。

在正常情况下，接收逻辑 RX 在进入到 Electrical Idle 状态之前，需要收到发送逻辑 TX 提供的 EIOS 序列，但是有时接收逻辑 RX 在没有收到 EIOS 序列时，发现 VRX-IDLE-DET-DIFFp-p 的值小于 75 mv 时，也可以进入 Electrical Idle 状态。PCIe 总线规范要求接收逻辑 RX 必须在 10 ms 之内判断出当前链路是否处于 Electrical Idle 状态。

接收逻辑 RX 除了可以使用差分电压逻辑，检测当前链路是否处于 Electrical Idle 状态之外，还可以通过其他方式推断出当前链路是否处于 Idle 状态。当 PCIe 链路处于不同的状态时，检测方法有所不同，如表 8–1 所示。值得注意的是，这种推断出的 Idle 状态并不等同于 Electrical Idle 状态。

表 8–1　接收逻辑 RX 推断当前链路是否处于 Idle 状态

链路状态	2.5 GT/s	5.0 GT/s
L0	在 128 μs 之内，接收逻辑 RX 没有收到 UpdateFC DLLP 或者 SKP 序列时，链路处于 Electrical Idle 状态	
Recovery. RcvrCfg	在 1280 个 UI 之内，接收逻辑 RX 没有收到 TS1 或者 TS2 序列	
Recovery. Speed Successful_speed_negotiation = 1	在 1280 个 UI 内，接收逻辑 RX 没有收到 TS1 或者 TS2 序列	
Recovery. Speed Successful_speed_negotiation = 0	在 2000 个 UI 之内，PCIe 链路没有退出 Electrical Idle 状态[⊖]	在 16000 个 UI 之内，PCIe 链路没有退出 Electrical Idle 状态
Loopback. active	在 128 μs 之内，PCIe 链路没有退出 Electrical Idle 状态	N/A

PCIe 总线通过上表判断而得出的 Idle 状态称为 Logical Idle 状态。在该表中出现的链路状态，如 L0、Loopback. active 等，将在下文详细解释。Electrical Idle 状态是 LTSSM 状态机的基础。Electrical Idle 状态是 PCIe 链路的相对静止状态，使用的功耗较低。当一个 PCIe 设备没有上电，处于复位状态或某些低功耗状态时，其使用的 PCIe 链路将处于 Electrical Idle 状态，读者需要进一步阅读下文的内容以详细了解 Electrical Idle 状态。

8.1.3　Receiver Detect 识别逻辑

Reciever Detect 识别逻辑的主要作用是检测对端的接收逻辑 RX 是否正常工作，Receiver Detect 识别逻辑是发送逻辑 TX 的一部分。PCIe 链路在初始状态时，需要检测对端设备是否

⊖ 没有退出 Electrical Idle 状态指接收逻辑 $V_{RX\text{-}IDLE\text{-}DET\text{-}DIFF_{p\text{-}p}}$的值小于 75 mV。

存在才能进行链路训练。Receiver Detect 识别逻辑的实现机理是通过检测对端设备接收逻辑的 DC 共模输入阻抗，来判断接收端是否存在。如果发送逻辑 TX 发现其负载的 DC 阻抗在 Z_{RX-DC} 范围之内或者小于 40 Ω 时，认为对端的接收逻辑 RX 存在。

PCIe 总线规范定义了接收逻辑 RX 在正常工作状态下的 DC 共模输入阻抗 Z_{RX-DC}，其值在 40 ~ 60 Ω 范围之内。当接收逻辑 RX 的 V_{cc} 没有上电，V_{D+} 信号或者 V_{D-} 信号的电压伏值大于 0 时，其 DC 共模输入阻抗 $Z_{RX-HIGH-IMP-DC-POS}$ 最小为 50 kΩ；当 V_{D+} 或者 V_{D-} 小于 0 时，其 DC 共模输入阻抗 $Z_{RX-HIGH-IMP-DC-NEG}$ 最小为 1.0 kΩ。

由此可见当对端接收逻辑 RX 可以正常工作时，其 DC 共模输入阻抗远小于“没有上电”时的状态。发送逻辑 TX 通过监控 V_{D+} 和 V_{D-} 信号的电压伏值，可以获得接收逻辑在正常工作状态和 V_{cc} 没有上电时的电流曲线，从而判断接收逻辑 RX 是否处于正常工作状态。在 PCIe 总线中，发送逻辑 TX 通过“发送 Detect 序列”判断对端接收逻辑 RX 是否存在。

PCIe 总线发送“Detect 序列”的原理是首先提高 V_{D+} 和 V_{D-} 信号的电压伏值，然后通过判断对端接收逻辑 RX 的阻抗变化，识别对端的接收逻辑 RX 是否正常工作。其具体实现方法如下所示。

（1）发送逻辑 TX 在提高 V_{D+} 和 V_{D-} 信号的电压伏值之前，需要保持 V_{D+} 和 V_{D-} 信号的伏值为一个恒定的 DC 共模电压值。

（2）发送逻辑 TX 暂时提高 V_{D+} 和 V_{D-} 信号的电压，但是其值不能超过原来共模电压伏值加上 $V_{TX-RCV-DETECT}$。此时发送端将产生一个脉冲波形至接收逻辑 RX。这个脉冲波形将穿越发送链路上的 AC 耦合电容，最后到达接收逻辑 RX。因为接收逻辑 RC 的 Z_{RX-DC} 较小，因此 $V_{TX-RCV-DETECT}$ 的值不能过大，否则将在接收逻辑 RX 上产生过大的电流㊀，从而可能损坏接收逻辑 RX。PCIe 总线规定 $V_{TX-RCV-DETECT}$ 的最大值为 600 mV。

（3）发送逻辑 TX 根据 V_{D+} 和 V_{D-} 信号的脉冲波形通过接收逻辑 RX 时的电流曲线，判断接收逻辑 RX 是否正常工作。如果通过这个电流曲线，发现是 Z_{RX-DC} 起作用，此时电流强度的有效值较大，表示接收逻辑 RX 正常工作；如果发现是 $Z_{RX-HIGH-IMP-DC-POS}$ 起作用，此时电流强度的有效值较小，表示接收逻辑 RX 不存在或者没有被加电。

值得注意的是，在 PCIe V2.x 规范中，并没有强行规定必须在 V_{D+} 和 V_{D-} 信号上都进行这种 Receiver Detect 测试。在有些实现上，可能仅使用 V_{D+} 或者 V_{D-} 信号进行这种 Receiver Detect 测试。在 LTSSM 状态机中，从 Detect 到 Polling 状态的切换时，需要使用 Receiver Detect 识别逻辑。

8.2 LTSSM 状态机

PCIe 总线在进行链路训练时，将使用 LTSSM 状态机。LTSSM 状态机由“Detect”、“Polling”、“Configuration”、“Disabled”、“Hot Reset”、“Loopback”、“L0”、“L0s”、“L1”、“L2”和“Recovery”共 11 个状态组成。这些状态分别与 PCIe 总线的链路训练、链路重训练、ASPM（Active State Power Management）和系统软件的电源管理相关。LTSSM 状态机的转换逻辑如图 8-5 所示，而各个状态的含义如下所示。

㊀ 在正常情况下，接收逻辑 RX 的 DC 共模电压为 0，Z_{RX-DC} 虽然较小也不会影响其正常工作。

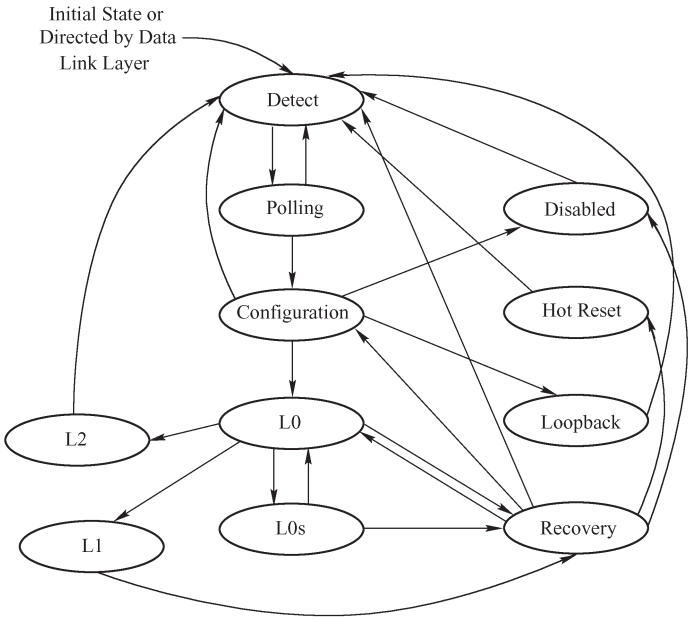

图 8-5　LTSSM 示意图

- Detect 状态。当 PCIe 链路被复位或者数据链路层通过填写某些寄存器后，LTSSM 将进入该状态。该状态也是 LTSSM 的初始状态。当 PCIe 链路处于该状态时，发送逻辑 TX 并不知道对端接收逻辑 RX 的存在，因此需要使用 Receiver Detect 识别逻辑判断对端接收逻辑 RX 是否可以正常工作，之后才能进入其他状态。
- Polling 状态。当 PCIe 链路进入该状态时，将向对端发送 TS1 和 TS2 序列，并接收对端的 TS1 和 TS2 序列，以确定 Bit/Symbol Lock、Lane 的极性。PCIe 链路处于该状态时，将进行 Loopback 测试，确定当前使用的 PCIe 链路可以正常工作。
- Configuration 状态。当 PCIe 链路进入该状态时，将确定 PCIe 链路的宽度、Link Number、Lane reversal、Polarity inversion 和 Lane-to-Lane 的延时。该状态是 LTSSM 状态机最重要的状态。值得注意的是 PCIe 链路在进行初始化时，链路两端统一使用 2.5GT/s 的数据传送率，直到进入 L0 状态。
- L0 状态。PCIe 链路的正常工作状态。PCIe 链路可以正常发送和接收 TLP、DLLP 和 PLP。PCIe 链路可以从该状态进入 Recovery 状态，以改变数据传送率。
- Recovery 状态。PCIe 链路需要进行链路重训练时需要进入该状态。该状态是 LTSSM 状态机的重要状态。
- L0s、L1 和 L2 状态。PCIe 链路的低功耗状态，其中 PCIe 链路处于 L0s 状态时，使用的功耗相对较高，而处于 L2 状态时，使用的功耗最低。
- Disabled 状态。系统软件可以通过设置寄存器，使 PCIe 链路进入 Disabled 状态。当 PCIe 链路的对端设备被拔出时，LTSSM 也需要进入该状态。
- Loopback 状态。PCIe 链路进入该状态时，发送端口将转发其接收端口接收到的数据，PCIe 测试仪器可以利用该状态进行数据测试。
- Hot Reset 状态。当处理器系统进行 Hot Reset 操作时，PCIe 链路将进入 Recovery 状

态，然后进入 Hot Reset 状态进行 PCIe 链路的重训练。

LTSSM 的各个状态将在下文详细介绍，本章将重点介绍 Detect、Polling、Configuration、L0 和 Recovery 状态，囿于篇幅并不对 Disabled、Loopback 和 Hot Reset 状态进行详细说明，对这些状态有兴趣的读者可以参阅 PCIe 总线规范，以获得进一步信息。

PCIe 设备的物理层进行复位后，LTSSM 状态机首先沿着“Detect”→“Polling”→“Configuration”→“L0”的路径进入到正常工作状态“L0”，这也是链路训练的正常工作路径，也是最重要的一条路径。

物理层链路训练所涉及的内容非常复杂。为便于读者理解，本书仅介绍其主要工作流程，即 PCIe 设备的正常工作路径，以帮助读者掌握链路训练的概要，而不会介绍异常处理和一些并不重要的分支。

8.2.1 Detect 状态

如图 8-6 所示，Detect 状态由 Detect.Quiet、Detect.Active 两个子状态组成。该状态的主要功能是检测 PCIe 链路上是否有 PCIe 设备存在，如果存在，一共使用了多少可用的 Lane 资源。在正常情况下，PCIe 链路将从 Detect 状态迁移到 Polling 状态。

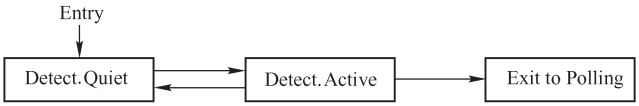

图 8-6 Detect 状态机

当 PCIe 设备进行传统复位操作[㊀]后，首先进入 Detect.Quiet 状态。在多数情况下，如果该 PCIe 设备的对端存在设备时，PCIe 链路的两端将同时进入 Detect.Quiet 状态。因为 PCIe 链路的两端可能同时进行复位操作。

在 PCIe 设备进入 Detect.Quiet 状态时，其发送逻辑 TX 处于“Electrical Idle”状态，此时该设备发送链路的 D+ 和 D- 信号的电压为 DC 共模电压，且为相同的值，此时发送逻辑 TX 使用的功耗最低。

值得注意的是，物理层也可以从 L2、Loopback、Disabled、Polling、Configuration 和 Recovery 状态进入 Detect. Quiet 状态，此时发送逻辑 TX 需要发送必要的 Idle 序列，通知对端设备的接收逻辑 RX，然后经过一段延时后才能进入“Electrical Idle”状态。而在 Detect 状态中，PCIe 设备的发送逻辑 TX 将直接进入到“Electrical Idle”状态，并不会使用 Idle 序列通知对端设备的接收逻辑 RX。

当 PCIe 设备处于 Detect.Quiet 状态时，缺省使用 2.5Gb/s 的数据传送率，即 PCIe V1.x 规定的数据传送率，并置 LinkUp、upconfigure_capable[㊁]等状态位为 0，此时数据链路层处于 DL_Inactive 状态。整个 PCIe 链路处于“完全静止”的状态。

当 PCIe 设备处于 Detect.Quiet 状态超过 12ms 之后，或者检测到 PCIe 链路上的任何一个

㊀ 不包括 FLR 方式，因为该复位并不影响 LTSSM 状态机。

㊁ upc onfigure_capable 状态位在 Configuration 状态收到“Link Upconfigure Capability”为 1 的 TS2 序列后将设置为 1，否则为 0。

Lane 退出“Electrical Idle”状态时，PCIe 设备将进入 Detect.Active 状态。

PCIe 设备进入 Detect.Active 状态后，其发送逻辑 TX 将向该链路的所有“未配置过的 Lane”端发送“Receiver Detection 序列”[⊖]，检测其对端的接收逻辑 RX 是否正常工作。如果所有 Lane 的接收逻辑 RX 都在正常工作时，PCIe 设备进入到 Polling 状态。

如果没有一个 Lane 的接收逻辑 RX 被检测到，PCIe 设备将进入到 Detect.Quiet 状态，此时可能 PCIe 链路的对端没有连接 PCIe 设备，或者该 PCIe 设备并没有正常工作。

如果仅有部分 Lane 正确检测到对端接收逻辑 RX 的存在，物理层将首先等待 12ms，然后使用该链路的所有“未检测成功过的 Lane”向对端重新发送“Receiver Detection 序列”，进一步识别可用的 Lane。如果这次的检测结果与第一次检测结果相同，物理层将这些“不可用”的 Lane 置为 Electrical Idle 状态，并进入 Polling 状态，如果两次结果不相同，则进入 Detect.Quiet 状态。

这些被标识为“Electrical Idle”状态的 Lane 将不会被 LTSSM 状态机继续使用。有时 PCIe 链路的两端虽然都连接 PCIe 设备，但是这些设备不一定利用了 PCIe 链路上所有的 Lane，下文将举例说明 Detect.Active 状态的运行机制。

假设一个 PCIe 链路由 8 个 Lane 组成，其上游端口与一个 Switch 连接，而下游端口连接一个 EP，如果这个 EP 仅使用了 2 个 Lane。那么第一次 PCIe 链路检测结果为 6 个 Lane 没有与 EP 进行连接，而经过 12ms 再次进行检测时，可能还检测到有 6 个 Lane 没有与 EP 进行连接，此时该 PCIe 链路认为 EP 仅使用了 2 个 Lane。如果第 2 次检查发现有 7 个或者 5 个 Lane 没有与 EP 进行连接，说明两次检测结果不一致，此时 PCIe 设备将要退回到 Detect.Quiet 状态，并择时重新进入 Detect.Active 状态，重新进行链路探测。

Detect 状态是 PCIe 设备进入的第一个状态，在这个状态中，PCIe 设备需要识别 PCIe 链路的拓扑结构。在这个状态中，物理层使用的检测手段都是基于 PCIe 链路的物理特性，只有 PCIe 设备发现 PCIe 链路的对端具有合法设备后，才能进入 Polling 状态。

8.2.2 Polling 状态

当 PCIe 设备在 Detect 状态中，识别完毕当前链路上可用的 Lane 资源之后，将进入 Polling 状态，Polling 状态由 Polling.Active、Polling.Compliance 和 Polling.Configuration 子状态组成。PCIe 设备可以从 Polling 状态进入到 Configuration 状态，继续进行链路训练，如果 PCIe 设备在 Polling 状态中出现某种错误时，将退回到 Detect 状态，重新进行 PCIe 链路的训练，Polling 状态机的转换逻辑如图 8-7 所示。

如该图所示，PCIe 设备将首先进入 Polling.Active 状态，然后进入 Polling. Configuration 状态，最后退出到 Configuration 状态。

PCIe 设备处于 Polling. Active 状态时，首先检查 Link Control 2 寄存器的“Enter Compliance Bit”位。如果该位为 1，PCIe 设备将进入 Polling.Compliance 状态，值得注意的是，PCIe 链路两端设备的“Enter Compliance Bit”需要被同时置 1，即 PCIe 链路两端的设备需要同时进入到 Polling.Compliance 状态。本节对 Polling. Compliance 状态不做进一步描述。

⊖ Receiver Dete ction 序列的发送方法见第 8.1.3 节。

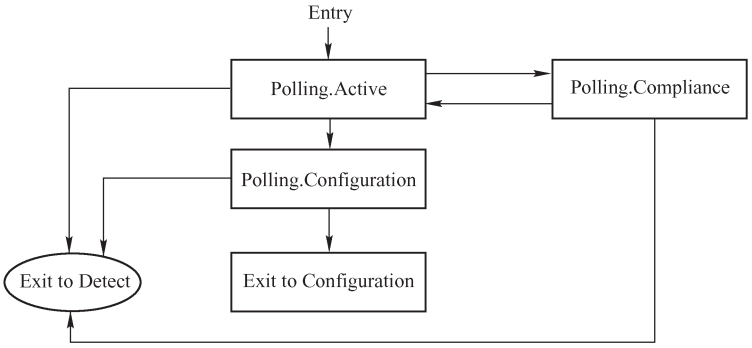

图 8-7　Polling 状态机

如果“Enter Compliance Bit”不为 1，则 PCIe 链路两端设备的发送逻辑 TX 需要向对端㊀至少发送 1024 个 TS1 序列，其中 TS1 序列的 Lane/Link Number 必须为“PAD”，即不设置 Lane/Link Number。PCIe 设备使用这些 TS1 序列，获得 Bit/Symbol Lock，这个过程大约需要 64 μs。值得注意的是，PCIe 链路两端设备退出 Detect 状态时，可能并不完全同步，因此两端设备交换 TS1 序列的过程也并不完全同步。

发送逻辑 TX 在发送 TS1 序列之前，需要保证 D + 和 D – 信号的 DC 共模电压恢复到正常工作值。因为发送逻辑 TX 在 Detect 状态进行“Receiver Detect”的过程中，曾经将 DC 共模电压提高了 $V_{TX\text{-}RCV\text{-}DETECT}$。

PCIe 设备在发送 1024 个 TS1 序列㊁的同时，如果其接收逻辑 RX 从全部“已被正确识别的 Lane”中收到了以下任意一种 8 个连续的报文序列后，该 PCIe 设备将进入 Polling. Configuration 状态。

（1）TS1 序列㊂，其 Lane/Link Number 为 PAD，而“Compliance Receive”位为 0。

（2）TS1 序列，其 Lane/Link Number 为 PAD，而“Loopback”位为 1㊃。值得注意的是，发送逻辑 TX 发送“Loopback”位为 1 的 TS1 序列后，PCIe 链路对端设备的接收逻辑 RX 将收到该序列，并将这个 TS1 序列使用内部 Loopback 逻辑直接回送给对端设备（并不是该设备重新生成的 TS1 序列），之后对端设备的接收逻辑 RX 将接收到之前发送逻辑 TX 发送的 TS1 序列。

（3）TS2 序列，其 Lane/Link Number 为 PAD。PCIe 链路两端设备进入的 LTSSM 状态机并不一定同步，可能对端 PCIe 设备可能已经进入了 Polling. Configuration 状态，此时该设备将向对端发送 TS2 序列，详见下文。

如果上述条件没有成立，PCIe 设备在经过 20ms 延时后，判断下列条件。如果这些条件同时成立时，PCIe 设备也将进入 Polling. Configuration 状态。

㊀ 此时发送逻辑 TX 使用在 Detect 状态中，已经正确识别的 Lane 发送这些 TS1 序列。

㊁ 这些 TS1 序列将被链路对端设备回传，因为此时 TS1 序列使用的 Loopback 位为 1。

㊂ 由于 PCIe 链路支持 Polarity Inversin，因此也可能收到 TS1 序列的补码。

㊃ PCIe 设备不可能从 Polling 状态进入 Loopback 状态，因此接收逻辑 RX 收到 Loopback 位为 1 的 TS1 序列并不会进入到 LTSSM 的 Loopback 状态。

（1）任何一个“已被正确识别的 Lane”收到了 8 个连续的 TS1 序列，其中 Lane/Link Number 为 PAD，而“Compliance Receive”位为 0 或者“Loopback”位为 1；或者收到 8 个连续的 TS2 序列。而且在收到第一个 TS1 序列之前，发送逻辑 TX 至少已经发送出 1024 个 TS1 序列。

（2）PCIe 设备从 Electrical Idle 状态中退出，并进入到 Polling. Active 状态时，所有“已被正确识别的 Lane”至少有一个 Lane 检测到对端设备。

如果该条件也没有成立，PCIe 设备将可能进入 Polling.Compliance 状态。如果 PCIe 链路上任何一个 Lane 的发送链路上连接了一个“对地阻抗为 50Ω”的电阻后，PCIe 设备也将强制进入 Polling.Compliance 状态。该状态的主要作用是对 PCIe 链路进行检测，本节对此不做进一步描述。

当 PCIe 设备进入 Polling.Configuration 状态时，物理层首先处理所有“已识别的”Lane 中，是否存在极性翻转（Lane Polarity Inversion）的现象，之后进行以下操作。

（1）置 Link Control 2 寄存器的“Transmit Margin”字段为 0b000。

（2）向对端“已识别的 Lane”连续发送 TS2 序列，其中 Link/Lane Number 为 PAD，而 Loopback 位为 0。

（3）当收到 8 个连续的 TS2 序列，而且一共收到 16 个 TS2 序列后，PCIe 设备将进入 Configuration 状态，否则经过 48ms 延时后进入 Detect 状态。

当 PCIe 链路两端设备收齐 TS2 序列后，将基本同步地进入 Configuration 状态。从这个角度来说，TS2 序列是为了同步“异步发送的 TS1 序列”。

在 Polling 状态机中，还有一个 Polling.Speed 子状态，该状态的主要作用是调整 PCIe 链路使用的数据传送率。当一个 PCIe 链路两端的设备可以支持高于 2. 5GT/s 的数据传送率时，可以首先进入该状态，改变 PCIe 链路的数据传送率。

在许多 PCIe 设备的具体实现中，并没有使用 Polling.Speed 子状态。此时在 PCIe 链路的训练过程中，将缺省使用 2. 5GT/s 的数据传送率。此时 LTSSM 状态机将首先沿着 Detect→Polling→Configuration→L0 的路径进入 L0 状态，并使用 2. 5GT/s 的数据传送率，即便 PCIe 链路两端的设备都支持更高的数据传送率。

当 PCIe 设备改变数据传送率时，需要在 L0 状态，通过向对端设备发送 TS1 序列（其 speed_change 位为 1），使 PCIe 链路两端设备进入 Recovery 状态后，才能改变缺省使用的数据传送率。

8. 2. 3　Configuration 状态

Configuratoin 状态是 LTSSM 的重要状态，该状态完成 PCIe 链路的主要配置工作，包括 Link Number 和 Lane Number 的协商，并使 PCIe 链路进入正常工作状态 L0。如图 8-8 所示，Configuration 状态由多个子状态组成。

其中 Configuration.Linkwidth.Start 和 Configuration.Linkwidth.Accept 状态判断当前 PCIe 链路的有效宽度；Configuration.Lanenum.Wait 和 Configuration.Lanenum.Accept 状态判断当前

PCIe 链路的物理“Lane Number”与逻辑“Lane Number”的对应关系；而 Configuration.Complete 状态负责处理“Lane-to-Lane de-skew”。由上文所述，一条 PCIe 链路可能由多个 Lane 组成，而使用这几个 Lane 进行数据传递时，可能存在速度差异，即“Skew”，“Lane-to-Lane de-skew”就是消除这个速度差异的方法。

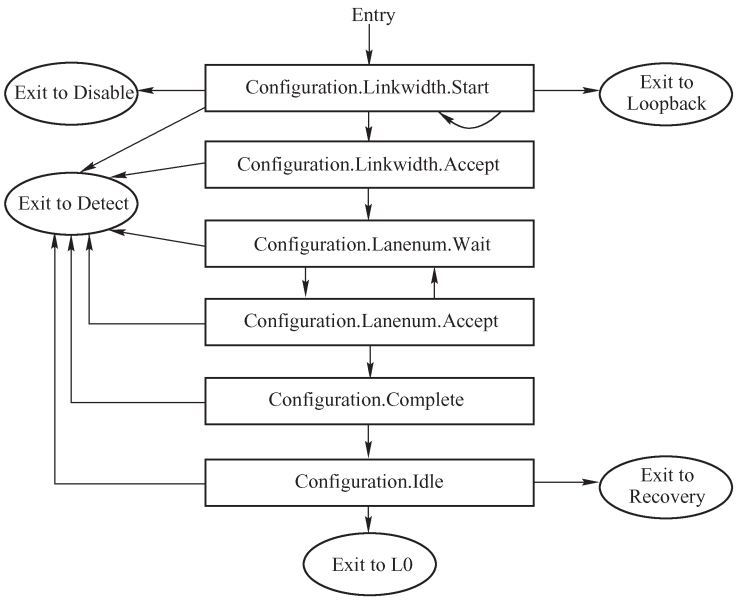

图 8-8　Configuration 状态机

在 LTSSM 状态机中，Polling 状态正常结束时将进入 Configuration 状态。此外当 Recovery 状态出现某些错误，没有正常进入 L0 状态时，也可能首先进入 Configuration 状态，然后经过错误处理之后，重新返回 L0 状态。

在 PCIe 总线中，LTSSM 状态机从 Polling 状态进入 Configuration 状态时 Linkup 位为 0，因为对应 Lane 不曾被激活；而从 Recovery 状态进入该状态时 Linkup 位为 1。进入 Configuration 状态时，PCIe 链路上游端口（包括 RC 端口或者 Switch 的下游端口）Link Status 寄存器的 Link Training 位被硬件置 1，从该状态进入 L0 状态时，该位被清零。

1. Link Number 的协商过程

PCIe 总线使用自协商的方法，确认不同 PCIe 链路的 Link Number。下文以一个具有两个端口的 Switch 为例，简要说明 Link Number 的协商机制，多端口 RC 的 Link Number 协商机制与此相似。该 Swith 的与 EP 的连接方法如图 8-9 所示。

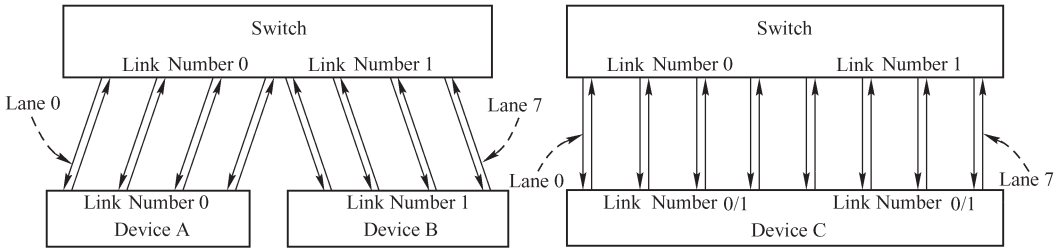

图 8-9　Link Number 的协商过程

在该 Switch 中共有 8 个 Lane，并且可以分解为 2 个 PCIe 链路[㊀]，每个链路的最大宽度为 4，即含有 4 个 Lane，这 4 个 Lane 并不能自由组合形成两个 PCIe 链路，而是第 0 ~ 3 个 Lane 组成一个 PCIe 链路，而第 4 ~ 7 个 Lane 组成另一个 PCIe 链路。其中每一个 PCIe 链路可以独立使用，连接不同的 EP，这些 EP 最多可以使用 4 个 Lane。当然这 8 个 Lane 也可以合并成一个 PCIe 链路，与一个 EP 连接。

Switch 不会预知其下游链路与 EP 的连接拓扑结构。因此在 Configuration. Linkwidth 状态时，Switch 将向第 0 ~ 3 个 Lane 发送 TS1 序列，其 Link Number 为 N；而向第 4 ~ 7 个 Lane 发送 TS1 序列，其 Link Number 为 N + 1。如图 8-1 所示，Device A 收到的 TS1 序列，其 Link Number 为 N，而 Device B 收到的 TS1 序列，其 Link Number 为 N + 1。Device A 或者 B 收到这些 TS1 序列后，将向 Switch 发送 TS1 序列，其 Link Number 为 N 或者 N + 1。

而 Device C 收到的 TS1 序列，其 Link Number 为 N 或者 N + 1。值得注意的是，此时 Device C 将选择唯一的 Link Number “或者为 N 或者为 N + 1” 回传给 Switch，而并不是将 “N 和 N + 1” 都回传给 Switch。

Switch 根据收到的 TS1 序列判断其链路的连接拓扑结构，如果 Switch 收到的 TS1 序列中包含两个不同的 Link Number，则表示 8 个 Lane 被分解为两个 PCIe 链路与两个 PCIe 设备相连；如果只有一个 Link Number，则表示 8 个 Lane 被合并为 1 个 PCIe 链路与 1 个 PCIe 设备相连。如果这个 Switch 支持 4 路 PCIe 链路，则 Switch 需要向这 4 个 PCIe 链路发送的 Link Number 为 N ~ N + 3，共 4 种 TS1 序列；如果 Switch 支持 8 路 PCIe 链路，则 Switch 需要向这 8 个 PCIe 链路发送的 Link Number 为 N ~ N + 7，共 8 种 TS1 序列。

PCIe 总线使用自协商的方法识别下游链路的拓扑结构。PCIe 总线在 Configuration 状态中，使用 Configuration. Linkwidth. Start 和 Configuration. Linkwidth. Accept 两个子状态，进行 PCIe 链路两端设备 Link Number 的协商，以确定下游链路与 PCIe 设备的连接拓扑结构。

当 PCIe 链路进入 Configuration. Linkwidth. Start 状态时，RC 端口或者 Switch 的下游端口将向其下游链路发送 TS1 序列以确定 PCIe 链路的 Link Number，下游设备也将会回送 TS1 序列。上文以连接 2 个 EP 的 Switch 为例，说明 Link Number 的自协商过程。RC 端口与其下游设备 Link Number 的协商过程与此类似，本节对此不做进一步说明。

当 PCIe 链路进入 Configuration 状态时，如果 Linkup 位为 0 时，Switch 需要向下游链路的每一个 Lane 都发送若干个 TS1 序列，在这个 TS1 序列中的 Link Number 字段分别为 0 和 1[㊁]，而 Lane Number 字段为 PAD。这里使用的 Link Number 由 Switch 内部由硬件逻辑进行编号，其编码在 0 ~ 255 之间，本节使用 0 和 1 作为 Link Number。注意在这个 TS1 序列中，Loopback 位不再为 1，否则对端设备将接收到的 TS1 序列直接 Loopback，回送给发送端。

值得注意的是，如果 Linkup 或者 upconfigure_capable 位为 1 时，即从 Recovery 状态进入该状态时，TS1 序列中的 Link Number 字段为 PAD（K23. 7），此时的处理方法与从 Polling 状态进入该状态有些区别。本节对这种情况不做进一步描述。

Switch 通过发送部件 TX，经过其下游链路发送完毕这些 TS1 序列后，开始监控其接收

㊀ 此时 Switch 中包含 2 个 LTSSM 状态机，也相应具有 2 个物理层、链路层和事务层。

㊁ 前 4 个 Lane 的 Link Number 为 0，而后 4 个 Lane 的 Link Number 为 1。

部件 RX。如果接收部件 RX 的任何一个 Lane 收到至少 1 个来自下游设备的 TS1 序列后（该序列的 Link Number 和 Lane Number 字段都为 PAD），需要进一步判断是否任何一个 Lane 收到 2 个连续的 TS1 序列（该序列的 Link Number 为 0 或者 1，即与发送的 TS1 序列 Link Number 相同，而 Lane Number 为 PAD）。上述检测成功之后，Switch 的下游端口将进入 Configuration.Linkwidth.Accept 状态。

而与 Switch 连接的下游设备在进入 Configuration.Linkwidth.Start 状态时，首先向其上游链路发送若干个 TS1 序列（该序列的 Link 和 Lane Number 字段都为 PAD），这也是 Switch 的下游端口首先收到这个 TS1 序列（Link 和 Lane Number 都为 PAD）的原因。

之后下游设备等待来自 Switch 的 TS1 序列，下游设备可能收到 TS1 序列中的 Link Number 为 0 或者 1（图 8-1 中的 Device C 可能收到这样的 TS1 序列），但是下游设备仅使用其中一个 Link Number（PCIe 总线并没有规定是使用 0 还是 1）组成若干个 TS1 序列，并通过其上游链路发送给 Switch，这个 TS1 序列的 Lane Number 为 PAD。

随后下游设备也进入 Configuration.Linkwidth.Accept 状态，至此在这条 PCIe 链路上的两个设备都将进入 Configuration.Linkwidth.Accept 状态。

在 Configuration.Linkwidth.Accept 状态时，Switch 的下游端口将分析接收到的 TS1 序列，并将所有 Link Number 相同的 Lane 组合在一起，通过此步骤，Switch 的下游端口可以确定下游链路的连接拓扑结构，即下游链路由哪些 Lane 组成。对于图 8-1 所示的实例，Device A 使用 Switch 下游链路的 Lane 0 ~ 3，Device B 使用 Switch 下游链路的 Lane 4 ~ 7，而 Device C 使用 Switch 下游链路的 Lane 0 ~ 7。

Switch 在确定了连接拓扑结构后，将向 PCIe 链路[㊀]的所有 Lane 发送若干个 TS1 序列，之后进入 Configuration.Lanenum.Wait 状态。值得注意的是，在这个 TS1 序列中 Link Number 为确定的值，而 Lane Number 在 0 ~ n - 1 之间，此时使用的 Lane Number 为逻辑号。在图 8-1 中，虽然 Device B 使用 Switch 的物理 Lane 为 4 ~ 7，但是发向物理 Lane 4 ~ 7 的 TS1 序列的 Lane Number 为 0 ~ 3。

在进入 Configuration.Linkwidth.Accept 状态时，下游设备等待 Switch 端口的 TS1 序列。如果下游设备收到 2 个连续的 TS1 序列，其 Link Number 等于在 Configuration.Linkwidth.Start 状态发送给 Switch 的 Link Number，而且 Lane Number 不为 PAD 时，下游设备将向 Switch 发送若干个 TS1 序列，之后下游设备进入 Configuration.Lanenum.Wait 状态。

该序列的 Link Number 为接收到的值，而 Lane Number 在 0 ~ m - 1[㊁]之间。如果下游设备没有使用错序连接方式，那么下游设备发送的 Lane Number 将与 Switch 的 Lane Number 直接对应；否则错序对应。本节虽然使用了一定篇幅讲述 PCIe 总线 Link Number 的自协商过程，但也仅说明了该过程的一个子集。PCIe 总线的自协商过程较为复杂，相对较难实现。

为此 PLX 公司采用了另外一种 Link Number 的协商机制。下面以 PEX8518 芯片[㊂]为例说明这种协商机制，该芯片并没有采用 PCIe 总线规定的协商机制，而是使用寄存器配置其下

㊀ 该 PCIe 链路可能由部分 Lane 组成，如与 Device A 连接的链路由 Lane 0 ~ 3 组成，与 Device B 连接的链路由 Lane 4 ~ 7 组成。

㊁ m≤n，因为下游设备可能只使用 PCIe 链路的部分 Lane。

㊂ PEX8518 是 PLX 公司设计的 Switch 芯片。

游端口的使用情况。

PEX8518 芯片的下游链路由 16 个 Lane 组成（分别为 0～15），最多可以支持 5 个端口，分别为 Port0～4。这 5 个端口能够使用的 Lane 由 Port Configuration 寄存器规定，而不是通过 Configuration 状态自适应检测其下游链路的拓扑结构。该寄存器的值与其下游端口的对应关系如表 8-2 所示。

表 8-2　Port Configuration 寄存器与下游端口的对应关系

Port Configuration 寄存器的值	链路宽度				
	端口 0	端口 1	端口 2	端口 3	端口 4
0x00	×4（0～3）	×4（4～7）	×4（8～11）	×4（12～15）	不使用
0x02	×8（0～7）	×8（8～15）	不使用	不使用	不使用
0x03	×8（0～7）	×4（8～11）	×4（12～15）	不使用	不使用
0x04	×8（0～7）	×4（8～11）	×2（12～13）	×2（14～15）	不使用
0x05	×8（0～7）	×2（8～9）	×2（10～11）	×4（12～15）	不使用
0x06	×8（0～7）	×2（8～9）	×4（10～13）	×2（14～15）	不使用
0x08	×8（0～7）	×2（8～9）	×2（10～11）	×2（12～13）	×2（14～15）
0x09	×4（0～3）	×4（4～7）	×4（8～11）	×2（12～13）	×2（14～15）

由上表所示，PEX8518 芯片使用静态表进行 Link Number 的确认，而没有使用 PCIe 总线规定的自协商机制，从而在保证配置灵活性的同时，极大简化了 Link Number 的协商难度。但是使用这种方法不适用于“下游链路拓扑结构不可预知”的应用。

因此在设计中，尽量避免使用 PEX8518 芯片连接一个 PCIe 插槽，因为这个插槽上的 PCIe 设备使用的 Lane 并不确定，使用静态分配 Lane 的方法并不合适，但是这并不妨碍 PEX8518 芯片在嵌入式领域，尤其在电信领域的大规模应用。

2. Lane Number 的协商过程

物理层在确认 PCIe 链路的 Link Number 之后，开始进行 Lane Number 的协商。PCIe 总线使用 Configuration.Lanenum.Wait 和 Configuration.Lanenum.Accept 两个子状态完成 Lane Number 的协商。下面仍然以图 8-9 为例说明 Lane Number 的协商过程。

与 Link Number 的协商过程相比，Lane Number 的协商过程较为简单。本小节以一个实例说明 Lane Number 的实现过程，而并不深究 LTSSM 状态机的详细迁移过程。

如图 8-1 所示，PCIe 链路允许错序连接，因此 Switch 的下游端口与下游设备的上游端口使用的 Lane Number 并不一定完全一致，因此物理层需要进行 Lane Number 的协商。当 Switch 与 Device A 完成 Link Number 的协商后，将进行 Lane Number 的协商。

首先 Switch 向 Device A 的 4 个 Lane 发送 TS1 序列，在这个 TS1 序列中的 Link Number 字段为 0，而 Lane Number 字段分别为 0、1、2 和 3。

当 Device A 收到这些 TS1 序列后，也将向 Switch 回送使用 TS1 序列，在这个 TS1 序列中的 Link Number 字段为 0，而 Lane Number 字段为 Device A 的物理 Lane Number 号。如果 Switch 与 Device A 使用图 8-1 左图所示的错序连接方法时，Device A 回送的 TS1 序列的 Link Number 字段为 3、2、1 和 0，如果使用图 8-1 中图所示的连接方法时，Device A 回送的 TS1 序列为 0、1、2 和 3。

当 Switch 收到 Device A 的回送 TS1 序列后，可以获得 PCIe 链路的连接信息，从而确定

当前 Lane 的连接拓扑结构，并完成逻辑 Lane Number 与物理 Lane Number 的对应关系。当使用图 8-1 左图所示的错序连接方法时，Switch 的逻辑 Lane Number 0 ~ 3 与 Device A 的物理 Lane Number 3 ~ 0 对应。

Switch 和 Device A 都可以处理 PCIe 链路的错序连接，本节所述的实例是 Switch 支持 PCIe 链路的错序连接而 Device A 不支持这种错序连接，此时 Device A 不进行物理 Lane Number 和逻辑 Lane Number 的转换，对于 Device A 而言，物理 Lane Number 等于逻辑 Lane Number。

当 Switch 的下游端口和对端设备的上游端口离开 Configuration.Lanenum.Accept 状态后，将进入 Configuration.Complete 状态进行 Link Number 和 Lane Number 的确认。

3. Link Number 与 Lane Number 的确认

PCIe 总线使用 Configuration.Complete 和 Configuration.Idle 状态进行 Link Number 与 Lane Number 的确认。在 Configuration.Complete 状态中，PCIe 链路的两端设备将使用 TS2 序列进行 Link 与 Lane Number 的确认。在这个 TS2 序列中的 Link Number 字段为之前确定的值，而 Lane Number 字段也为已经确认的号码。

当链路两端的设备收到这些 TS2 序列后，将进一步消除 PCIe 链路不同 Lane 的漂移（De-skew），并设置合理的 N_FTS 的值，还有一个重要的操作是记录当前 PCIe 设备能够支持的数据传送率[㊀]，然后进入 Configuration.Idle 状态。

PCIe 设备处于 Configuration.Idle 状态时，PCIe 链路已经设置完毕，此时将向对端至少发送 16 个 Idle 序列，当接收逻辑 RX 收到这些 Idle 序列后，将置 LinkUp 状态位为 0b1，PCIe 数据链路层将从 DL_Inactive 状态迁移到 DL_Init 状态，而物理层将进入正常工作状态 L0。

8.2.4 Recovery 状态

Recovery 状态是 LTSSM 状态机的重要状态，其复杂程度超过 Configuration 状态。该状态可以从 L0、L1 和 L0s 状态进入，当 PCIe 设备进入低功耗状态，需要进行链路重训练时，将经过 Recovery 状态之后，才能重新进入正常工作状态，第 8.3 节将讲述如何从 L1 和 L0s 状态进入 Recovery 状态。

Recovery 状态机如图 8-10 所示，该状态机由 Recovery.RcvrLock、Recovery.Speed、Recovery.RcvrCfg 和 Recovery.Idle 共四个子状态组成。

1. 从 L0 状态进入到 Recovery 状态

当 PCIe 设备工作在 L0 状态时，出现以下情况时，将进入 Recovery 状态。

（1）更改数据传送率

PCIe 设备需要改变数据传送率时，将进入 Recovery 状态。在 Configuration 状态中，PCIe 链路两端设备记录了“当前 PCIe 设备能够支持的数据传送率”，如果两端设备都支持大于 2.5GT/s 的数据传送率，可以从 L0 状态进入 Recovery 状态。

在 PCIe 设备中存在两个状态位。其中 directed_speed_change 位为 1 时，表示 PCIe 设备

㊀ 如果 PCIe 设备需要改变数据传送率时，可以在 Recovery 状态中进行更改，但是在 Configuration 状态不能进行该操作。

希望更改数据传送率，而 changed_speed_recovery 位为 1 时表示 PCIe 链路已经完成数据传送率的更改。

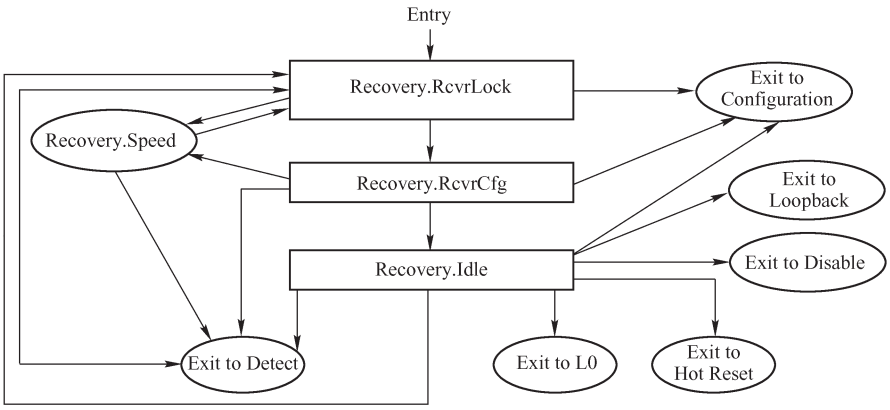

图 8-10　Recovery 状态机

当数据链路层处于 DL_Active 状态时，系统软件可以置 directed_speed_change 位为 1，同时复位 changed_speed_recovery 位为 0，随后该设备将进入 Recovery 状态。值得注意的是 PCIe 链路的两端设备需要同时改变 directed_speed_change 和 changed_speed_recovery 位。

（2）更改链路宽度

PCIe 设备在 Configuration 状态时，如果设置 upconfigure_capable 状态位为 1，可以更改链路宽度。此时该设备需要从 L0 状态进入 Recovery 状态，之后才能进行该操作。

（3）已配置完毕的 Lane 中收到 TS1 或者 TS2 序列。

（4）检测或者推断出对端设备处于 Idle 状态。

当 PCIe 设备检测到对端处于 Idle 状态时，将有可能进入 Recovery 状态。此处的 Idle 状态由两部分内容组成，一个通过硬件检测对端设备的发送逻辑 TX 是否处于“Electrical Idle”状态；另一个通过逻辑推断对端设备的发送逻辑是否处于 Logical Idle 状态。

在正常情况下，发送逻辑 TX 在进入 Electrical Idle 状态时，需要发送 EIOS 序列。如果对端的接收逻辑 RX 检测到 PCIe 链路实际上已经进入 Idle 状态，但是并没有收到 EIOS 序列时，将认为 PCIe 链路出现某种故障，PCIe 总线并没有规定在这种情况下，PCIe 设备是进入 Recovery 状态还是继续保持在 L0 状态中，但是在多数实现中，都将进入 Recovery 状态。

值得注意的是，一个 PCIe 设备进入 Recovery 状态之后，将向对端发送 TS1 序列，而对端设备收到这个 TS1 序列后，也将从 L0 状态进入 Recovery 状态。因此在 PCIe 链路中，当一端设备进入 Recovery 状态后，对端设备也将进入该状态。

2. Recovery.RcvrLock 状态

PCIe 设备进入 Recovery 状态时，将首先到达 Recovery.RcvrLock 状态。PCIe 设备进入该状态时将连续向对端“已配置完毕的 Lane”发送 TS1 序列，该序列的 Link 和 Lane Number 为之前在 Configuration 状态中设置的值。

如果 directed_speed_change 位为 1 时，发送的 TS1 序列的 speed_change 位也将设置为 1。PCIe 设备如果在 Recovery.RcvrLock 状态时，收到 speed_change 位为 1 的 TS1 序列时，该设

备的 directed_speed_change 位也将改变为 1。

下文以一个实例说明 speed_change 位和 directed_speed_change 位的作用。假设在一条 PCIe 链路上连接了两个设备，EP A 和 Switch 的下游端口 B，而且这两个设备的工作状态的初始值为 L0。

如果 EP A 希望进入 Recovery 状态改变数据传送率，则置 directed_speed_change 位为 1，随后进入 Recovery. RcvLock 状态，同时向端口 B 发送 speed_change 位为 1 的 TS1 序列。

而当端口 B 处于 L0 状态时，收到这个 TS1 序列后，并不会检测 speed_change 位，而直接进入 Recovery 状态。然后向 EP A 发送若干个 speed_change 位为 0 的 TS1 序列，因为此时端口 B 的 directed_speed_change 位为 0。

当端口 B 收到“8 个连续”的从 EP A 发来的“speed_change 位为 1”的 TS1 序列（还包括 EP A 支持的数据传送率种类）之后，directed_speed_change 位将置为 1，然后再向 EP A 发送“speed_change 位为 1”的 TS1 序列。值得注意的是，在上述实例中，端口 B 在 Recovery.RcvLock 状态将发送两种不同的 TS1 序列，一种序列的 speed_change 位为 0，而另一种序列的 speed_change 位为 1。

PCIe 设备连续发送 TS1 序列的同时，将通过其接收逻辑 RX 检测，是否收到 8 个连续的 TS1 或者 TS2 序列，是否这些序列使用的 Link 和 Lane Number 与该设备发送的值相同，而且这些序列的 speed_change 位是否与 directed_speed_change 位相同。如果检测成功，该设备将进入 Recovery.RcvrCfg 状态。在该状态中，PCIe 设备将重新获得 Bit/Symbol Lock，并处理不同 Lane 之间的 Skew，这也是该状态位被命名为“Lock”的原因。

PCIe 设备还可以从该状态直接进入 Recovery.Speed BFQ状态，如果当前 PCIe 链路已经工作在大于 2. 5GT/s 的数据传送率，但是并不能稳定工作（并没有正确获得 Bit/Symbol Lock），此时需要进入 Recovery.Speed 状态，将数据传送率更改为 2. 5GT/s。PCIe 设备还可以从该状态进入 Configuration 或者 Detect 状态。本节对此不做详细介绍。

3. Recovery.RcvrCfg 状态

PCIe 设备进入 Recovery.RcvrCfg 状态后，将向对端发送 TS2 序列，该序列的 Link 和 Lane Number 为之前在 Configuration 状态设置的值。如果 directed_speed_change 位为 1 时，该序列的 speed_change 位也必须为 1。发送这些 TS2 序列的主要目的是进一步确认 Recovery.RcvrLock 状态使用 TS1 序列所获得的结果。

之后 PCIe 设备通过其接收逻辑 RX 检测，是否收到了 8 个连续的 TS2 序列，这些序列使用的 Link 和 Lane Number 与该设备发送的值是否相同，而且这些序列的 speed_change 位是否与 directed_speed_change 位相同。

如果检测成功，PCIe 设备将根据 TS2 序列中的 speed_change 位决定进入 Recovery.Speed 状态，还是 Recovery.Idle 状态。

如果该 PCIe 设备能够支持的数据传送率与接收的 TS2 序列中的数据传送率重合，比如都支持 5GT/s 的数据传送率，而且 speed_change 位为 1 时，将进入 Recovery.Speed 状态，PCIe 设备从 Recovery.RcvrCfg 状态迁移到 Recovery.Speed 状态还有一个补充条件，即接收到第一个 TS2 序列后，至少向对端发送 32 个连续的 TS2 序列。

如果 Speed_change 位为 0 时，将进入 Recovery.Idle 状态，PCIe 设备从 Recovery.RcvrCfg 状态迁移到 Recovery.Idle 状态还有一个补充条件，即从接收到第一个 TS2 序列后，向对端发

送 16 个连续的 TS2 序列。

如果在 Recovery.RcvrLock 状态中，PCIe 设备没有处理不同 Lane 之间的 Skew，在该状态中，PCIe 设备可以处理不同 Lane 之间的 Skew。PCIe 设备还可以从该状态进入 Configuration 或者 Detect 状态。本节对此不做详细介绍。

4. Recovery.Speed 状态

PCIe 设备进入该状态时，如果使用的数据传送率为 2.5GT/s，则其发送逻辑 TX 需要发送 1 个 EIOS 序列，然后进入 Electrical Idle 状态；如果使用的数据传送率为 5.0GT/s，则其发送逻辑 TX 需要发送 2 个 EIOS 序列，然后进入 Electrical Idle 状态。此时该 PCIe 设备需要等待其接收逻辑 RX 也进入 Electrical Idle 状态，才能进一步工作。接收逻辑检测对端发送逻辑 TX 是否处于 Electrical Idle 状态，使用的方法如第 8.1.2 节所示。

当 PCIe 设备的发送与接收逻辑都进入到 Electrical Idle 状态后，至少需要等待 800ns 判断 PCIe 链路两端是否成功完成数据传送率的协商，或者等待 1ms 判断协商是否成功。协商成功后 successful_speed_negotiation 位被置为 1，否则将置 0。

如果协商成功，PCIe 设备将置 directed_speed_change 位为 0，然后从该状态回到 Recovery.RcvrLock 状态。值得注意的是在 Recovery.Speed 状态并不发送 PLP 检查新的数据传送率是否能够使用。而在 Recovery.RcvrLock 状态中，通过 TS1 序列判断对端设备是否能够获得 Bit/Symbol Lock，确定新的数据传送率是否正常工作。如果不能正常工作，该设备还将从 Recovery.RcvrLock 状态回到 Recovery.Speed 状态，将数据传送率降低为 2.5GT/s 后，再次回到 Recovery.RcvrLock 状态。

5. Recovery.Idle 状态

PCIe 链路进入 Recovery.Idle 状态后，将连续向对端发送 Idle 序列，当接收逻辑 RX 的所有可用 Lane 收到 8 个连续的 Idle 序列，而且发送逻辑 TX 发送了 16 个 Idle 序列后，将进入 L0 状态，完成链路训练或者重训练过程；否则将视情况进入 Disabled、Loopback、Hot Reset、Configuration 和 Detect 状态，本节对 Recovery.Idle 进入这些状态不做详细分析。

8.2.5 LTSSM 的其他状态

如图 8-5 所示，在 LTSSM 中还含有 Disabled、Host Reset、Loopback、Recovery、L0、L1、L2 和 L0s 状态。其中 L0、L1、L2 和 L0s 与 PCIe 的电源管理相关，在第 8.3 节将详细介绍这些状态的转换关系。

PCIe 链路还可以从 Configuration 和 Recovery 状态进入 Disabled 状态。当系统软件需要关闭 PCIe 链路时，可以通过设置 Link Control 寄存器的 Link Disabled 位，使 PCIe 链路进入 Disabled 状态。物理层进入 Disabled 状态时，将禁止 PCIe 链路的使用，然后视情况进入 PCIe 链路的初始状态 Detect，重新进行 PCIe 链路的训练工作。

而 Loopback 状态是一种调试状态，PCIe 总线测试仪器可以使 PCIe 链路的对端设备进入该状态，然后对 PCIe 链路进行测试。

Hot Reset 状态从 Recovery 状态进入，当系统软件对 PCIe 链路进行 Hot Reset 操作时，PCIe 链路将进入该状态，然后进入 PCIe 链路的初始状态 Detect。

8.3 PCIe 总线的 ASPM

PCIe 总线的电源管理包含 ASPM 和软件电源管理两方面内容。所谓 ASPM 是指 PCIe 链路在没有系统软件参与的情况下，由 PCIe 链路自发进行的电源管理方式。而软件电源管理指 PCI PM 机制，PCIe 总线的软件电源管理与 PCI 总线兼容。

对于一个通用处理器系统，电源管理的硬件实现与软件处理过程都较为复杂。而对于一个专用的处理器系统，如手机应用，在多数情况下，更侧重于在某些用户场景中，功耗的使用情况，其设计难度相对较小。而无论是对于通用还是专用处理器系统，电源管理都需要处理器与外部设备的参与，协调完成。

PCIe 总线为 PCIe 设备提供了几种低功耗模式。在一个处理器系统中，PCIe 设备的低功耗模式需要与处理器的低功耗模式协调工作，以最优化整个处理器系统的功耗。

对于某些专用的处理器系统，外部设备之间也需要协调工作，以最优化整个处理器系统的使用的功耗。目前电源管理已经成为处理器系统实现的一个热点。但是 PCI/PCIe 总线提供的电源管理机制远非完美，该管理机制仅考虑了外部设备与处理器系统之间的电源使用关系，并没有考虑外部设备之间电源的使用关系。

在一个处理器系统中，外部设备越来越智能化，处理器与外设间的通信基本等同于两个处理器间的通信，适用于这类处理器系统的电源管理机制也有待研究。在有些处理器系统中，专门设置了用于电源管理的微处理器，协调“主处理器”与“外部设备”及外部设备间的电源使用情况，以最优化整个处理器系统的电源管理。目前在智能手机的设计中，通常具有专门用于电源管理的微处理器。

8.3.1 与电源管理相关的链路状态

PCIe 总线定义了一系列与电源管理相关的链路状态。

- L0 状态。PCIe 设备的正常工作状态。
- L0s 状态。PCIe 设备处于低功耗状态。系统软件不能控制 L0 状态和 L0s 状态间的迁移过程，这两个状态的迁移只能由 ASPM 控制。
- L1 状态。PCIe 设备使用的功耗低于处于 L0s 状态时的功耗。
- L2/L3 Ready 状态。PCI 设备进入 L2 或者 L3 状态之前使用的过渡状态。
- L2 状态。PCIe 设备仅使用辅助电源工作，主电源已经被关闭。在 PCIe 总线中 L1 和 L2 状态是可选的。
- L3 状态。该状态也被称为“Link off”状态，此时 PCIe 设备使用的 Vcc 电源已经被关闭。
- LDn 状态。该状态是一个“伪”状态，PCIe 链路处于 L2、L3 状态时，需要通过 LDn 状态之后才能进入 L0 状态。该状态由 LTSSM 状态机的 Detect、Polling 和 Configuration 等状态组成。

这些与电源管理相关的状态机迁移模型如图 8-11 所示。

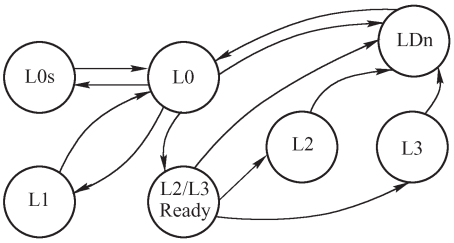

图 8-11　电源管理状态机

本节重点说明 L0、L0s 和 L1 状态的工作原理以及如何使用 ASPM 机制进行状态迁移。在第 8.4 节将讲述系统软件如何设置寄存器使 PCIe 设备进入 L0、L0s 和 L1 状态。

在 PCIe 设备中，Link Capabilities 寄存器的 ASPM Support 字段表示当前 PCIe 设备可以支持的链路状态，该字段只读。而 Link Control 寄存器的 ASPM Control 字段为可读写的，PCIe 设备根据 ASPM Support 字段判断当前 PCIe 链路是否支持 L0s 和 L1 状态，还是同时支持这两种状态，并设置 ASPM Control 字段。

8.3.2　L0 状态

PCIe 设备从 Configuration、Recovery 和 L0s 状态进入 L0 状态时，L0 状态是 PCIe 设备的正常工作状态。此时 PCIe 设备可以通过 PCIe 链路，发送和接收 TLP、DLLP 和 PLP，此时 LinkUp 状态位为 1，而数据链路层处于 DL_Active 状态。PCIe 设备从 L0 状态可以进入 Recovery、L0s、L1 和 L2/L3 Ready 状态，本节重点讨论从 L0 状态进入 Recovery、L0s 和 L1 状态的情况。而从 L0 状态进入 Recovery 状态的条件见第 8.2.4 节。

1. 进入 L0s 状态

当 PCIe 设备发现链路“空闲”时[⊖]，可以主动进入 L0s 状态。RC、EP 和 Switch 进入 L0s 状态的“空闲”条件并不相同。对于含有多个 Function 的上游端口，只有所有 Function 都处于“空闲”状态时，才被认为“空闲”。此处“空闲”的定义与 LTSSM 的 Electrical Idle和 Logical Idle 间没有任何联系。

对于 RC 或者 EP 的端口，当以下两个条件同时满足时，当前端口被认为是“空闲”的。

（1）没有准备发送的 TLP，或者对端没有提供足够的 Credit[⊜]。

（2）没有准备发送的 DLLP。

对于 Switch 的上游端口，当以下三个条件同时满足时，发送逻辑 TX 认为 PCIe 链路是“空闲”的。

（1）Switch 所有下游端口的接收链路处于 L0s 状态。

（2）没有准备发送的 TLP，或者对端没有提供足够的 Credit。

（3）没有准备发送的 DLLP。

⊖ 当空闲时间不大于 7μs 时，将进入 L0s 状态，但是 PCIe 总线并没有规定进入 L0s 状态的最小时间。

⊜ 对端没有提供足够的 Credit 时，发送端不能发送 TLP。

对于 Switch 的下游端口，当以下三个条件同时满足时，发送逻辑 TX 认为 PCIe 链路是“空闲”的。

（1）Switch 的所有上游端口的接收链路处于 L0s 状态。

（2）没有准备发送的 TLP，或者对端没有提供足够的 Credit。

（3）没有准备发送的 DLLP。

值得注意的是，一个接收端或者发送端的发送逻辑 TX 和接收逻辑 RX 在同一时刻，可能处于不同的 LTSSM 状态，一个处于 L0，而另一个处于 L0s。

2. 进入 L1 状态和 L2 状态

PCIe 设备可以通过上层软件，将链路状态从 L0 状态迁移到 L1 或者 L2 状态。当 PCIe 设备进入 D1 ~ D3 状态时，其上游链路将进入 L1 状态；而进入 $D3_{Cold}$状态时，其上游链路将进入 L2 状态。D1 ~ D3 和 $D3_{Cold}$状态的详细说明见第 8.4.1 节。而 PCIe 链路的两端设备需要同时进入 L1 或者 L2 状态。

首先两端设备的发送逻辑 TX 分别向对端发送 EIOS 序列，随后发送逻辑 TX 进入 Electrical Idle 状态。如果 PCIe 设备的接收逻辑 RX 从任意 Lane 中收到 1 个或者 2 个 EIOS 序列后，该设备将进入到 L1 或者 L2 状态。当 PCIe 链路工作在 2.5GT/s 时，需要发送 1 个 EIOS 序列；如果工作在 5GT/s 时，需要发送 2 个 EIOS 序列。

值得注意的是，PCIe 设备处于 L1 状态和 L2 状态时，D + 和 D - 信号输出的 DC 共模电压并不相同。处于 L1 状态时，PCIe 设备的发送逻辑 TX 仍然需要维持一个相对较低的 DC 共模电压，其伏值比其处于 L0 状态时低 VTX - CM - DC - ACTIVE - IDLE - DELTA（最小值为 0，最大值为 100mv），如公式 8 - 1 所示。

$$| V_{TX-CM-CD[DuringL0]} - V_{TX-CM-Idle-DC[During\ Electrical\ Idle]} | \leqslant 100\ \text{mV} \qquad (8-1)$$

而 PCIe 设备在 L2 状态时，其发送逻辑 TX 并没有这种限制，而且其发送逻辑和接收逻辑基本处于下电状态。这正是 PCIe 设备处于 L2 状态时使用的功耗低于 L1 状态的原因，同时也是从 L2 状态进入 L0 状态，更加耗时的原因。

如果 PCIe 设备支持 Beacon 机制，那么处于 L2 状态时，PCIe 设备的发送逻辑 TX 能够发送 Beacon 信号，而接收逻辑 RX 能够检测 Beacon 信号[㊀]。Beacon 机制是一种唤醒机制。当 PCIe 设备处于 L2 状态时，可以使用该机制唤醒。

8.3.3 L0s 状态

PCIe 设备必须支持 L0s 状态。L0s 状态是一个低功耗状态，PCIe 设备进入或者退出该状态不需要系统软件的干预，其状态转换由硬件控制完成。L0s 的状态转换由两部分组成，一个是接收状态机，另一个是发送状态机。

同一个 PCIe 设备的发送逻辑 TX 和接收逻辑 RX，在同一时刻可能处于不同的链路状态，其中一个为 L0，而另一个为 L0s。例如当一个 EP 进行 DMA 写操作时，其发送逻辑 TX 一直被使用，因此处于 L0 状态，而接收逻辑 RX 可能长时间没有被使用，从而可以暂时处

㊀ Beacon 信号并不是实际信号，而是通过 D + 和 D - 信号发送的一个频率在 30KHz ~ 500MHz 之间的脉冲信号，该信号可以唤醒 PCIe 设备。在许多 PCIe 设备中，并不含有 WAKE#信号，此时需要使用 Beacon 机制唤醒 PCIe 设备。

于 L0s 状态，以降低功耗。

1. 发送逻辑 TX 状态机

L0s 的发送状态机如图 8-12 所示，该状态机由 Tx_L0s.Entry、Tx_L0s.Idle 和 Tx_L0s.FTS 状态组成。

图 8-12　L0s 的发送状态机

PCIe 设备处于 L0 状态发现链路为临时“空闲”状态时，将进入 Tx_L0s.Entry 状态。处于该状态时，发送逻辑 TX 首先向对端发送 1 或者 2 个 EIOS 序列[㊀]，之后进入 Electrical Idle 状态。再经过 20ns 延时后，发送逻辑 TX 进入 Tx_L0s.Idle 状态。

当发送逻辑 TX 处于 Tx_L0s.Idle 状态时，如果 PCIe 设备需要发送数据报文，发送逻辑 TX 将退出 Tx_L0s.Idle 状态，进入 Tx_L0s.FTS 状态。发送逻辑 TX 处于 Tx_L0s.FTS 状态时，向对端顺序发送 N_FTS 个 FTS 序列[㊁]和 1 个 SKP 序列之后，将进入 L0 状态。

2. 接收逻辑 RX 状态机

L0s 的接收状态机如图 8-13 所示，该状态机由 Rx_L0s.Entry、Rx_L0s.Idle 和 Rx_L0s.FTS 状态组成。PCIe 设备可以从 L0s 状态进入 L0 或者 Recovery 状态。

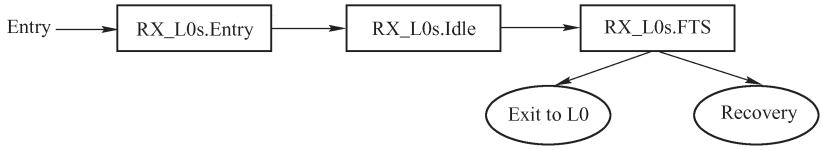

图 8-13　L0s 的接收状态机

接收逻辑 RX 处于 L0 状态时，如果收到 1 个 EIOS 序列后，将进入 Rx_L0s.Entry 状态。接收逻辑 RX 在 Rx_L0s.Entry 状态经过一段延时[㊂]后，将进入 Rx_L0s.Idle 状态。

接收逻辑 RX 在 Rx_L0s.Idle 状态中将持续监测接收链路，如果发现对端设备的发送逻辑 TX 退出“Electrical Idle”状态时，接收逻辑 RX 将进入 Rx_L0s.FTS 状态。

当接收逻辑 RX 处于 Rx_L0s.FTS 状态时，PCIe 链路的每一个 Lane 都将收到 N_FTS 个 FTS 序列，接收逻辑 RX 使用这些 FTS 序列重新获得 Bit/Symbol Lock。如果对端发送逻辑 TX 发送的 FTS 序列不足，接收逻辑 RX 将无法成功获得 Bit/Symbol Lock，此时 PCIe 设备将进入 Recovery 状态。

当接收逻辑 RX 收到足够数量的 FTS 序列，又收到了一个 SKP 序列后（该 SKP 序列的作用是 De-Skew），将从 Rx_L0s.FTS 状态迁移到 L0 状态。

8.3.4　L1 状态

L1 状态是一个比 L0s 状态使用功耗更低的状态，PCIe 设备从 L1 状态恢复到 L0 状态，比 L0s 状态恢复到 L0 状态的延时更长。PCIe 设备进入或者退出该状态可以不需要系统软件

㊀ 当前 PCIe 链路为 2.5GT/s 时发送 1 个 EIOS 序列，否则发送 2 个 EIOS 序列。

㊁ 如果 Link Control 寄存器的 Extended Sync 位为 1 时，发送部件将向对端发送 4096 个 FTS 序列。

㊂ PCIe 规范规定这段延时为 $T_{TX\text{-}IDLE\text{-}MIN}$（最小值为 20 ns）。

的干预。当然系统软件也可以通过设置某些寄存器，使 PCIe 链路的两端设备同时进入 L1 状态。在 PCIe 总线中，L1 状态是一个可选状态。

其中只有下游设备（EP 或者 Switch 的上游端口）可以主动“进入 L1 状态”，而上游设备（RC 或者 Switch 的下游端口）必须与下游设备进行协商后才能进入 L1 状态。当下游设备满足以下条件时，可以进入 L1 状态。

（1）PCIe 设备支持 L1 状态。

（2）PCIe 设备没有准备发送的 TLP 和 DLLP。

（3）如果下游设备是一个 Switch，这个 Switch 的所有下游端口处于 L1 或者更高一级的节电状态。

而上游设备需要经过协商才能进入 L1 状态。

（1）首先下游设备向上游设备发送 PM_ Active_State_Request_L1 报文。

（2）上游设备收到这个 DLLP 报文后，如果该上游设备可以进入 L1 状态，则向下游设备发送 PM_ Request_Ack 报文；如果不能进入，则发送 PM_Active_State_Nak 报文。

在 PCIe 总线中，L1 状态由 L1.Entry 和 L1.Idle 两个子状态组成，如图 8-14 所示。

图 8-14　L1 状态机

PCIe 设备从 L0 状态首先进入 L1.Entry 状态。PCIe 设备处于 L1.Entry 状态时，发送逻辑 TX 处于 Electrical Idle 状态。PCIe 设备在此状态停留 20ns 后，进入 L1.Idle 状态。接收逻辑在 L1.Idle 状态中将持续监测接收链路，如果发现其对端发送逻辑 TX 退出“Electrical Idle”状态时，将从 L1.Idle 状态首先迁移到 Recovery 状态，而不是 L0 状态。

PCIe 设备处于该状态时，其发送逻辑 TX 可以处于高阻抗或者低阻抗模式，而其接收逻辑 RX 必须处于低阻抗模式。

8.3.5　L2 状态

当 PCIe 设备处于 L2 状态时，使用的功耗低于 L1 状态，但是恢复到 L0 状态的延时更长。当 PCIe 设备处于 L2 状态时，需要首先迁移到 Detect 状态，重新进行链路训练。L2 状态是一个可选状态。L2 状态机由 L2.Idle 和 L2.TransmitWake 两个子状态组成，如图 8-15 所示。

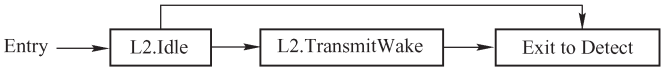

图 8-15　L2 状态机

在 L2.Idle 状态中，接收逻辑 RX 的端接必须处于低阻抗模式，而发送逻辑 TX 必须在 Electrical Idle 状态中至少停留 20ns。当一个 EP、Switch 或者 RC 的某个端口被唤醒后，将首先从 L2.Idle 状态迁移到 L2.TransmitWake 状态。

PCIe 设备进入 L2.TransmitWake 状态后，将向 RC 端口或者 Switch 的下游端口发送 Beacon 信号。当 RC 端口收到这个 Beacon 信号后，将进入 Detect 状态进行链路训练；而当 Switch 收到 Beacon 信号被唤醒后，其上游端口将进入 L2.TransmitWake 状态，并向上游链路转发这个 Beacon 信号，并逐级唤醒 PCIe 链路的上游设备。

当 EP 或者 Switch 上游端口的接收逻辑 RX，发现其对端发送逻辑 TX 退出 Electrical Idle

状态时，将从 L2.TransmitWake 状态迁移到 Detect 状态，重新进行链路训练。

8.4 PCI PM 机制

PCIe 总线与 PCI 总线使用的 PCI－PM 管理机制兼容。在 PCIe 设备的扩展配置空间中定义了 Power Management Capabilities 结构，该结构中含有一系列寄存器，这些寄存器的详细说明见第 4.3.1 节。

系统软件通过修改 PMCSR 寄存器的 Power State 字段，可以使 PCIe 设备进入不同的节能状态 D-State，如 D0、D1、D2 和 D3 状态。其中 D0 是正常工作状态，功耗最高，而 D1、D2 和 D3 为低功耗状态。其中 D1 的休眠等级最低，功耗相对较高，而 D3 的休眠等级最高，功耗相对较低。D－State 的状态转换关系如图 8－16 所示。

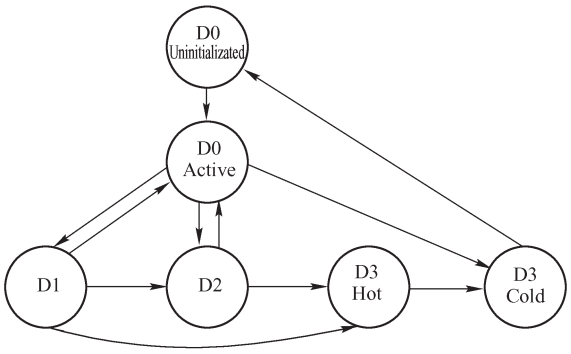

图 8－16　D-State 状态机

值得注意的是，当 PCIe 设备进行状态迁移时，PCIe 链路也需要视情况进入相应的状态，如进入 L1 或者 L2 等状态。

8.4.1 PCIe 设备的 D-State

PCIe 设备的 D-State 由 D0、D1、D2 和 D3 状态组成。其中 D0 状态由 $D0_{uninitializated}$ 和 $D0_{active}$ 两个子状态组成，而 D3 状态由 $D3_{hot}$ 和 $D3_{cold}$ 两个子状态组成。

1. D0 状态

PCIe 设备必须支持 D0 状态，该状态由 $D0_{uninitializated}$ 和 $D0_{active}$ 两个子状态组成。当 PCIe 设备处于 $D0_{uninitializated}$ 状态时，该 PCIe 设备并没有被系统软件使能，此时该 PCIe 设备仅能接收配置读写请求 TLP，不能主动发出其他 TLP。此时该 PCIe 设备配置寄存器的 Command 寄存器为复位值 0x00。此时虽然 PCIe 设备已经被加电，但是并不能正常使用。

当 PCIe 设备处于 $D0_{active}$ 状态时，PCIe 设备处于正常工作模式，并没有任何节电措施。但是 PCIe 设备仍然可以使用 ASPM 机制，将链路状态迁移到 L0s 或者 L1 状态，以降低功耗。值得注意的是，ASPM 机制与 PCI PM 机制是独立的。

当 PCIe 设备进行复位后，该设备将首先进入 $D0_{uninitializated}$ 状态。系统软件通过修改 PMCSR 寄存器的 Power State 字段，也可以使设备从 D3hot 状态迁移到该状态。值得注意的是 D3cold 状态迁移到该状态的过程与复位操作等效。当系统软件改写 Command 寄存器的状态

位使能 PCIe 设备后，该设备从 $D0_{uninitializated}$迁移到 $D0_{active}$状态。

2. D1 和 D2 状态

D1 和 D2 状态分别为 PCIe 设备的轻度和重度休眠状态。这两个状态为 PCIe 设备的可选状态，PCIe 设备处于 D1 状态时的功耗高于 D2 状态。

PCIe 设备处于这两个状态时，除了 PME 消息之外，不能主动发送其他 TLP；除了接收配置请求 TLP 外，不能接收其他 TLP。当 PCIe 设备处于这两种状态时，可以向 RC 发送 PME 消息，通知系统软件该 PCIe 设备进入休眠状态。当 PCIe 设备进入 D1 或者 D2 状态时，PCIe 链路将进入 L1 状态。PCIe 设备可以从 D1 和 D2 状态直接返回到 $D0_{active}$状态。

3. D3 状态

PCIe 设备必须支持 D3 状态，D3 状态由 $D3_{hot}$和 $D3_{cold}$两个子状态组成。PCIe 设备处于 $D3_{hot}$状态与处于 D1/D2 状态时的功能类似，只是 PCIe 设备只能从 $D3_{hot}$状态返回 $D0_{uninitializated}$状态，而不能返回 $D0_{active}$状态。对于 PCIe 设备，从 $D3_{hot}$状态返回 $D0_{uninitializated}$状态的过程相当于热复位。

当 PCIe 设备的 Vcc 电源被移除时，PCIe 设备无论处于何种状态，都将进入 $D3_{cold}$状态。值得注意的是一个 PCIe 设备使用两种电源 V_{cc}和 V_{aux}，V_{cc}电源被移除并不意味着 PCIe 设备被完全下电。

有些 PCIe 设备在处于 $D3_{cold}$状态时仍然可以发出 PME 消息，此时这个 PCIe 设备负责发送 PME 消息的功能模块必须使用 V_{aux}而不是 V_{cc}进行供电。

8.4.2 D-State 的状态迁移

如图 8-16 所示，PCIe 设备可以进行 D-State 的状态迁移。大多数 D-State 的状态迁移都是系统软件通过修改 PMCSR 寄存器的 Power State 字段实现的，但是仍然有些状态迁移采用了其他方式。

- 使能 Command 寄存器的命令位，可以使设备从 $D0_{uninitializated}$状态迁移到 $D0_{active}$状态。
- PCIe 设备的 V_{cc}被移除时，$D3_{hot}$状态将迁移到 $D3_{cold}$状态。
- 当 PCIe 被唤醒，V_{cc}重新上电之后，PCIe 设备将从 $D3_{cold}$状态迁移到 $D0_{uninitializated}$状态。

当 PCIe 设备进行 D-State 状态迁移时，PCIe 链路的状态也可能随之变化。PCIe 设备的 D-State 状态与 PCIe 链路状态的对应关系如表 8-3 所示。

表 8-3 D-State 状态与 PCIe 链路状态的对应关系

下游设备的 D-State	上游设备可能的 D-State	可能的链路状态
D0	D0	L0，L0s，L1，L2/L3 Ready
D1	D0 ~ D1	L1，L2/L3 Ready
D2	D0 ~ D2	L1，L2/L3 Ready
$D3_{hot}$	D0 ~ $D3_{hot}$	L1，L2/L3 Ready
$D3_{cold}$	D0 ~ $D3_{cold}$	L2，L3

由上表可以发现，上游设备所处的 D-State 等级小于或等于下游设备的休眠等级。如下游设备处于 D1 状态时，上游设备不能处于比 D1 更高的休眠等级，如 D2 或者 D3 状态。

当设备处于 D0 ~ $D3_{cold}$状态时，ASPM 机制可以根据链路的使用情况进行链路状态的迁移，而无需软件的干预。下文以 PCIe 设备从 D0 迁移到 D1 说明 D-State 进行状态迁移时，

PCIe 链路如何进行状态迁移。

当系统软件修改 PMCSR 寄存器的 Power State 字段，将 PCIe 设备从 D0 迁移到 D1 状态时，上游设备与下游设备将协调工作，完成 PCIe 设备的状态切换，并改变链路的状态，其实现过程如图 8-17 所示。

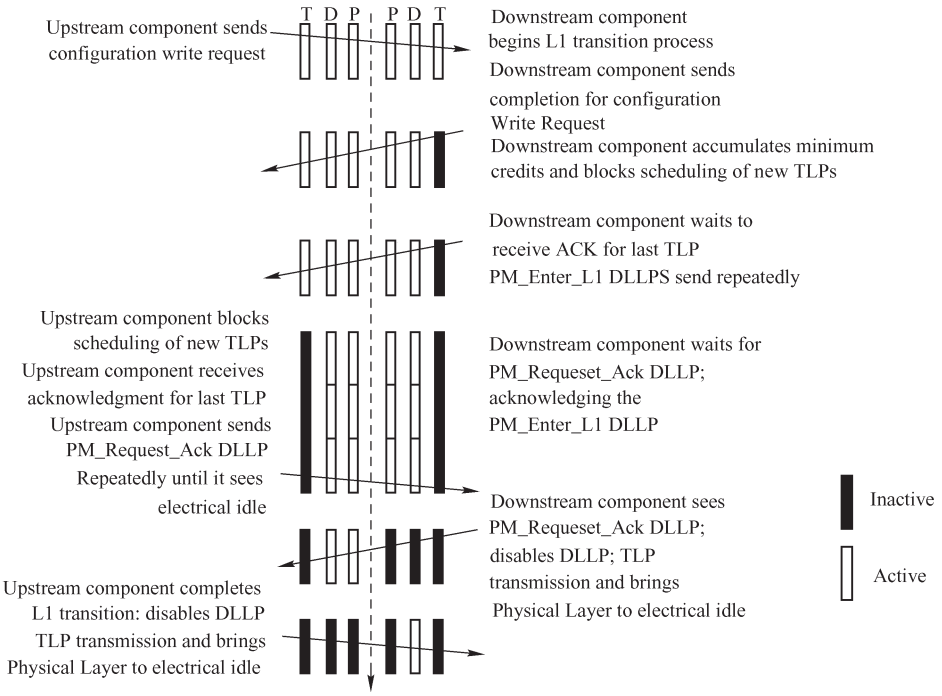

图 8-17　PCIe 设备从 D0 到 D1 的状态迁移

（1）上游设备向下游设备发送配置写请求，改变下游设备 PMCSR 寄存器的 Power State 字段，从而使下游设备从 D0 状态迁移到 D1 状态。

（2）下游设备收到这个配置写请求 TLP 后，将改变 PMCSR 寄存器的 Power State 字段，并向上游设备发送配置写完成 TLP。这个配置写完成 TLP 首先需要经过数据链路层，并从对端获得足够的发送 Credit[⊖]后，将这个配置写完成 TLP 通过数据链路层发送到对端。

（3）下游设备的事务层收到数据链路层的确认后，得知配置写完成 TLP 已经被上游设备正确接收后（详见 ACK/NAK 协议），将挂起下游设备的事务层。并向上游设备连续发送 PM_Enter_L1 DLLP，同时等待来自上游设备的 PM_Request_Ack 报文。

（4）上游设备收到下游设备的 PM_Enter_L1 DLLP 后，首先禁止发送新的 TLP，并等待之前发送的 Non-Post TLP 得到确认后，挂起上游设备的事务层，并向下游设备连续发送 PM_Request_Ack DLLP。

（5）下游设备在没有收到上游设备的 PM_Request_Ack DLLP 之前，虽然事务层已经被挂起，但是数据链路层和物理层仍然可以正常工作，此时数据链路层可以正确接收来自上游

⊖ 发送 Credit 的详细说明见第 9 章流量控制。

端口的 DLLP，并发送 ACK/NAK 和流量控制相关的一些 DLLP。

（6）当下游设备收到 PM_Request_Ack DLLP 后，将停止发送 PM_Enter_L1 DLLP，挂起数据链路层，然后将物理层置为 Electrical Idle 状态。

（7）上游设备发现其接收链路处于 Electrical Idle 状态时，将停止发送 PM_Request_Ack DLLP，并挂起数据链路层，然后将物理层置为 Electrical Idle 状态。此时 PCIe 链路将进入 L1 状态。

当 PCIe 链路处于 L1 状态时，如果系统软件需要改变下游 PCIe 设备 PMCSR 寄存器的 Power State 字段，PCIe 链路需要首先从 L1 状态迁移到正常工作状态 L0，下游设备才能接收这个配置写请求 TLP。

PCIe 设备其他 D-State 状态的迁移过程与此大同小异，详见 PCIe 总线规范，本节对此不做进一步描述。

8.5 小结

本章重点介绍 PCIe 总线的 LTSSM 状态机，该状态机的迁移模型较为复杂。本章仅介绍了该状态机的基本工作路径，对此部分有兴趣的读者可以阅读 PCIe 总线规范，进一步了解相关内容。

LTSSM 状态机在 PCIe 总线规范中处于核心地位，深入理解该状态机的运转模型，有利于底层软件工程师深入理解 PCIe 设备的工作状态，从而开发出质量较高的程序。

本章还使用一定篇幅介绍了 PCIe 总线的电源管理模型。目前电源管理已经成为计算机体系结构的热点。一个合理的电源管理模型需要软硬件的共同参与，但是硬件设计仍然决定了电源管理模型的节电效率。