



PCIe 科普教程

古猫 著

2017.6



目录

1. 0 PCIe 概述	4
2. 0 Transaction layer 事务层概述	7
2. 1 TLP 的前世今生	9
2. 2 TLP 事务处理方式	12
2. 3 TLP 结构解析	16
2. 4 Flow Control 机制概述	21
2. 5 Flow Control 缓存架构及信用积分	22
2. 6 Flow Control 初始化	23
2. 7 Flow Control 的实现过程	28
2. 8 事务排序机制	32
3. 0 数据链路层概述	35
3. 1 数据链路层 DLLP 结构及类型	36
3. 2 数据链路层 Ack/Nak 机制	40
4. 0 物理层结构解析	54
4. 1 物理层数据流解析	59
5. 0 PCIe 总线电源管理	65
6. 0 PCIe 系统复位方式	72
7. 0 PCIe 热插拔	78



微信公众号平台：PCIe 专题文章列表 (点击即可跳转)

[PCIe 系列专题之一：PCIe 技术概述](#)

[PCIe 系列专题之二：2.0 Transaction layer 事务层概述](#)

[PCIe 系列专题之二：2.1 TLP 的前世今生](#)

[PCIe 系列专题之二：2.2 TLP 事务处理方式解析](#)

[PCIe 系列专题之二：2.3 TLP 结构解析](#)

[PCIe 系列专题之二：2.4 Flow Control 机制概述](#)

[PCIe 系列专题之二：2.5 Flow Control 缓存架构及信用积分](#)

[PCIe 系列专题之二：2.6 Flow Control 初始化](#)

[PCIe 系列专题之二：2.7 Flow Control 的实现过程](#)

[PCIe 系列专题之二：2.8 事务排序机制](#)

[PCIe 系列专题之三：3.0 数据链路层概述](#)

[PCIe 系列专题之三：3.1 数据链路层 DLLP 结构及类型](#)

[PCIe 系列专题之三：3.2 数据链路层 Ack/Nak 机制解析](#)

[PCIe 系列专题之四：4.0 物理层结构解析](#)

[PCIe 系列专题之四：4.1 物理层数据流解析](#)

[PCIe 系列专题之五：PCIe 总线电源管理](#)

[PCIe 系列专题之六：PCIe 系统复位方式](#)

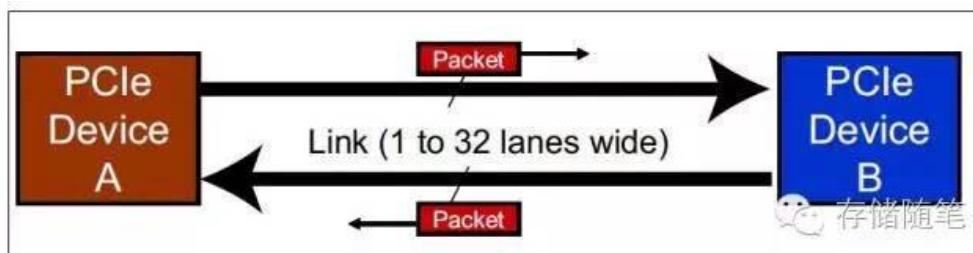
[PCIe 系列专题之七：PCIe 热插拔](#)



1.0 PCIe 概述

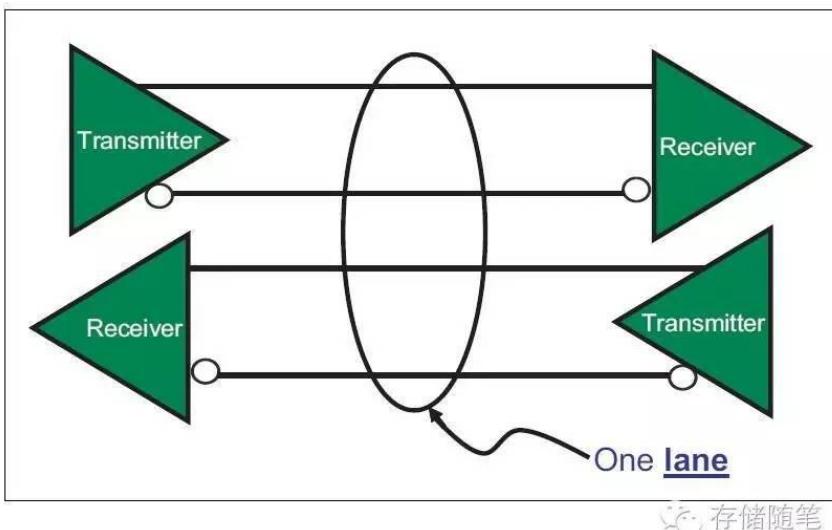
SSD 的协议标准除了 SATA，还有一个更先进的协议标准，就是 PCIe。PCIe 总线使用了高速差分总线，并采用了端到端的连接方式。

两个设备之间的的传输通道，称为 Link，由 1, 2, 4, 8, 16, 32 个 Lane 组成。Lane 的数目代表 Link 的传输宽度 (x1, x2, x4, x8, x16, x32)。

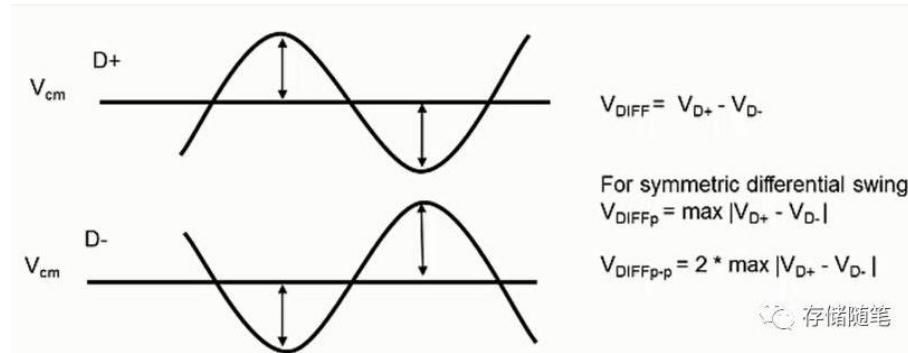


Lane 又是何方神圣呢？

Lane 是发送端与接收端之间的一个传输回路。由两组差分信号组成。如下图，



扩展： 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，能有效抑制电磁干扰 EMI(Electro Magnetic Interference)。

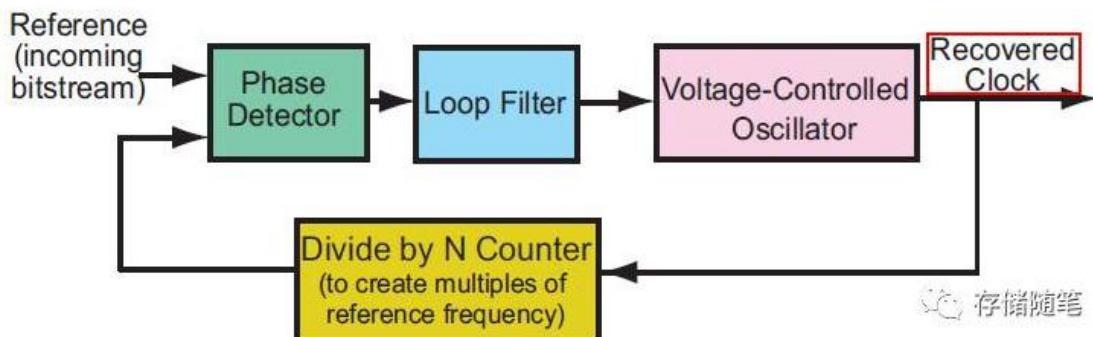


目前正式发布的最新仍是第三代 PCIe 技术。不过，第四代 PCIe 协议预计在今年 6 月会的 PCIe 开发者大会上正式发布，相信很快也会跟大家见面了。我们这里就先列出 PCIe Gen1/2/3 的传输速率对比如下表：

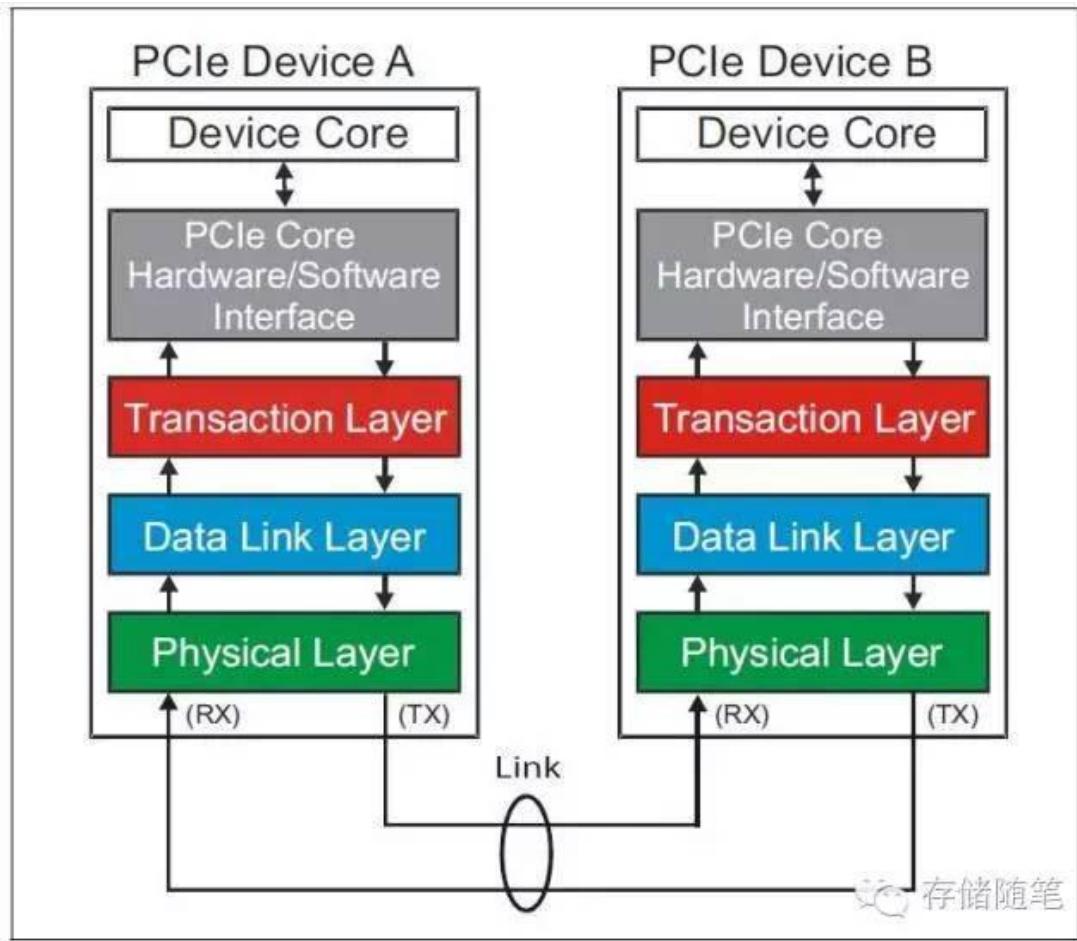
Link Width	x1	x2	x4	x8	x12	x16	x32
Gen1 Bandwidth (GB /s)	0.5	1	2	4	6	8	16
Gen2 Bandwidth (GB/s)	1	2	4	8	12	16	32
Gen3 Bandwidth (GB/s)	2	4	8	16	24	32	64

这里需要提一下：Gen1/Gen2 采用的是 8b/10b 的编码，而 Gen3 则采用的是 128b/130b 的编码。

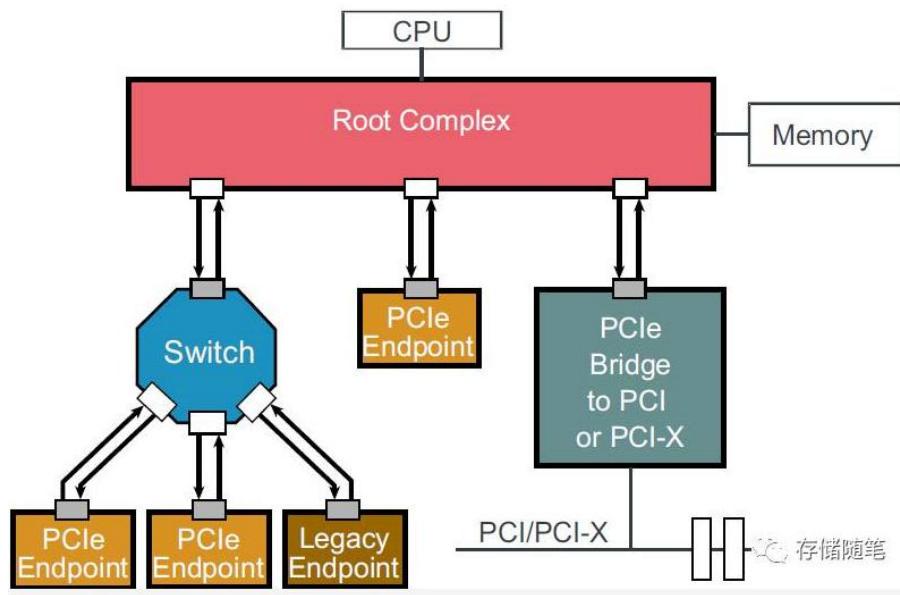
PCIe 总线物理链路间的数据传送使用基于时钟的同步传送机制，但是在物理链路上并没有时钟线，PCIe 总线的接收端会通过 PLL 锁相环从接收报文中提取接收时钟，从而进行同步数据传递。



PCIe 是一种封装分层协议，主要包括事务层（Transaction layer），数据链路层（Data link layer）和物理层（Physical layer）。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、数据链路层和事务层，并最终到达 Device Core。



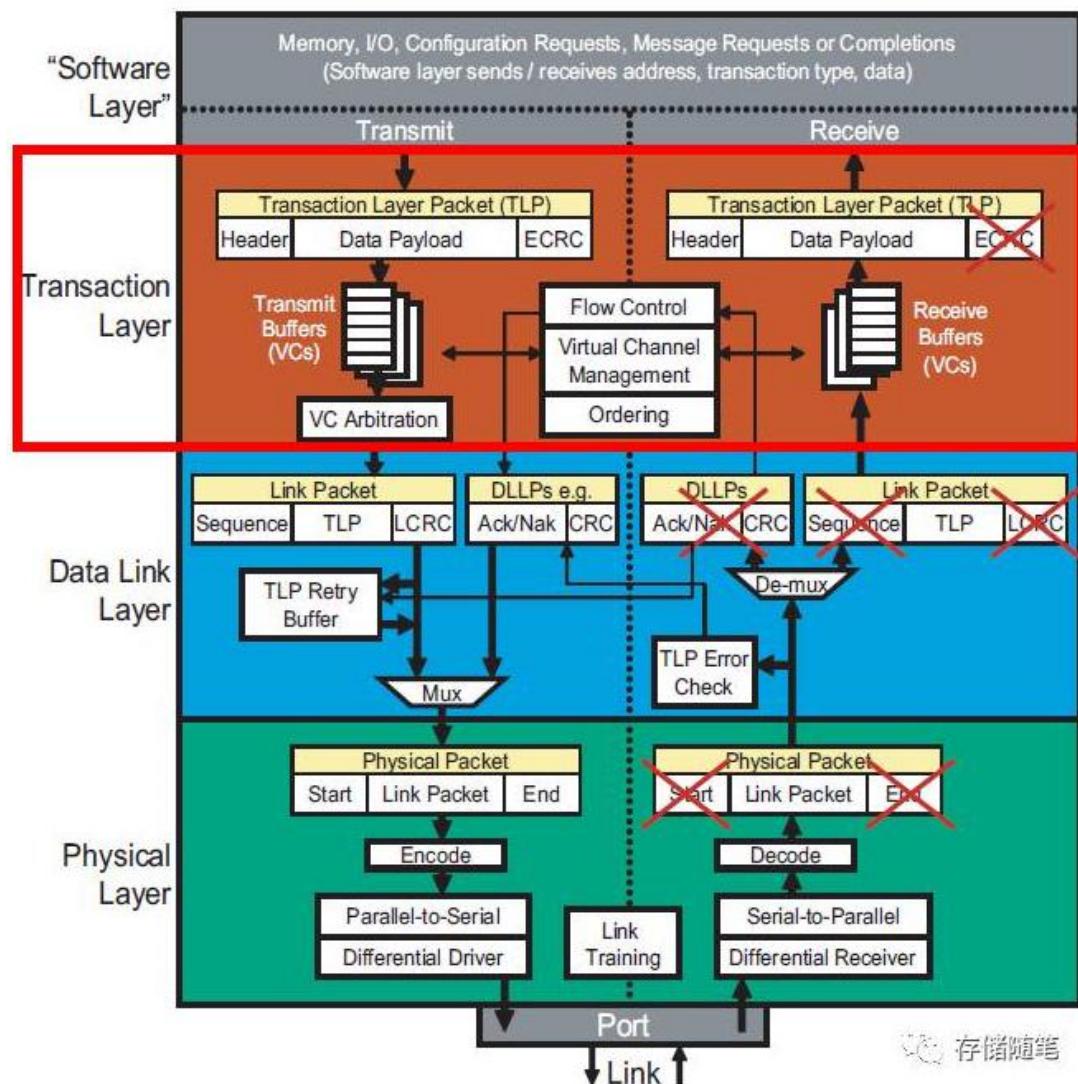
在一条 PCIe 链路中，这两个端口是完全对等的，分别连接发送与接收设备，而且一个 PCIe 链路的一端只能连接一个发送设备或者接收设备。因此 PCIe 链路必须使用 **Switch** 扩展 PCIe 链路后，才能连接多个设备。





2.0 Transaction layer 事务层概述

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、数据链路层和事务层，并最终到达 Device Core。



事务层的主要职责可以概述为：

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 (Memory Read/Write)**、**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**。



Transaction Type	Non-Posted or Posted
Memory Read	Non-posted
Memory Write	Posted
Memory Read Lock	Non-posted
IO Read	Non-posted
IO Write	Non-posted
Configuration Read (Type 0 and 1)	Non-posted
Configuration Write (Type 0 and 1)	Non-posted
Message	Posted
AtomicOp	Non-Posted

扩展： PCIe 中有两大类总线事务：Non-Posted 和 Posted：

a, **Non-Posted:** 需要 completion 返回响应包；

b, **Posted:** 不需要 completion 返回响应包.

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。

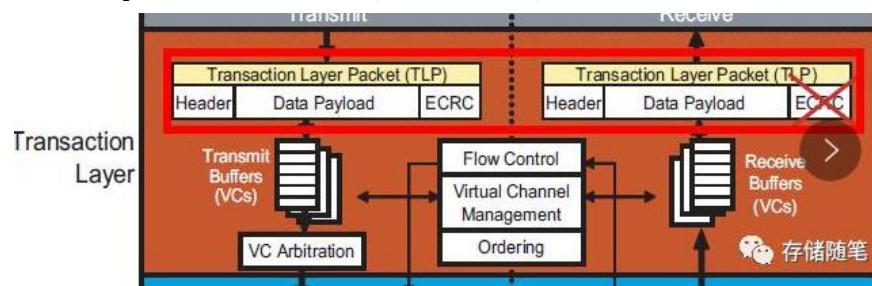
PCIe 的事务层还支持流量控制(Flow control)和虚通路管理(Virtual Channel Management)等一系列特性，而 PCI 总线并不支持这些新的特性。

在 PCIe 总线中，不同的总线事务采用的路由方式不相同。PCIe 总线使用的数据报文首先在事务层中形成，这个数据报文也被称之为事务层数据报文，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。

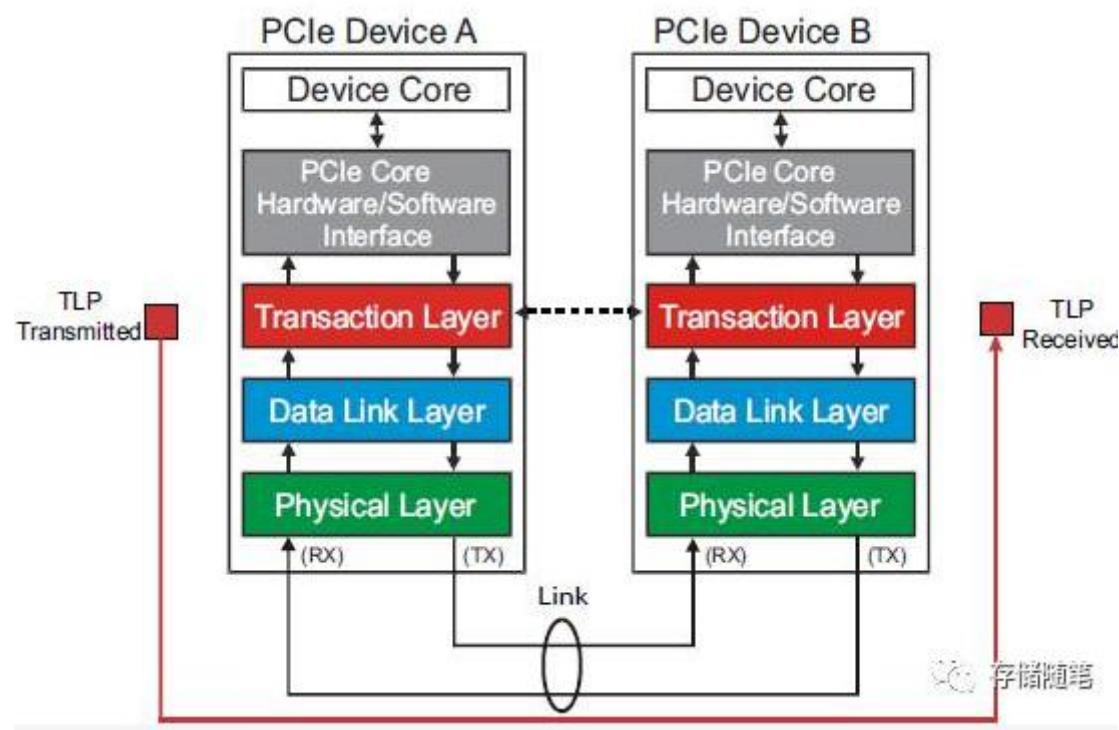


2.1 TLP 的前世今生

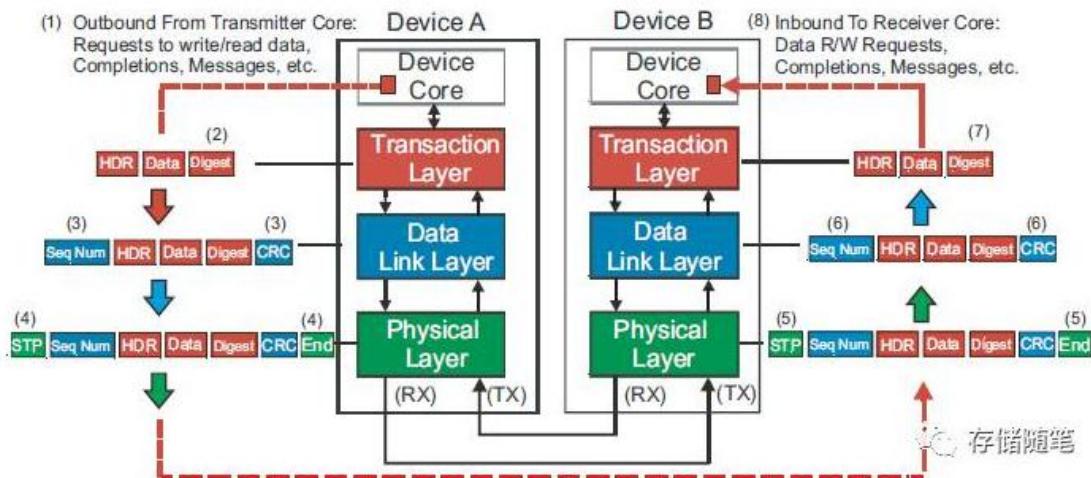
PCIe 总线使用的数据报文首先在事务层中形成，这个数据报文也被称之为事务层数据报文，即 TLP(Transaction Layer Packet)，TLP 在经过数据链路层时被加上 Sequence Number 前缀和 CRC 后缀，然后发向物理层。



生活中，有时，我们会陷入一个哲学性的思考：“我们来自哪里，终归何方？”同样，TLP 也有这个命题的解答。TLP 来自发送设备的事务层，历经“磨难”，终归接收端的事务层。

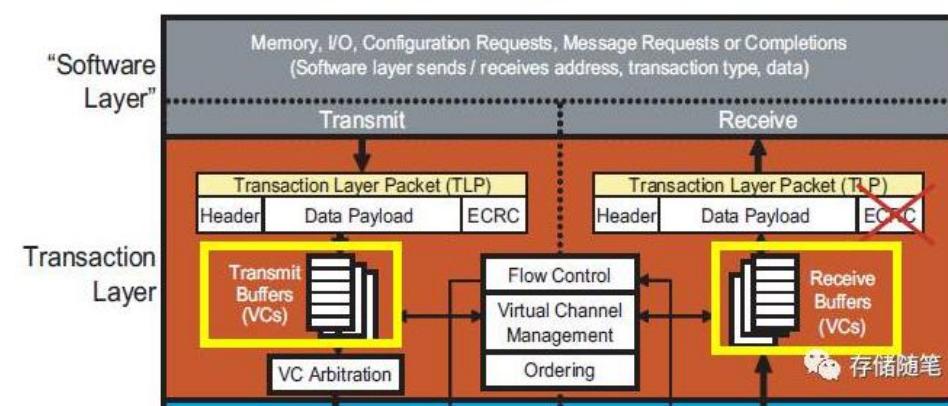


那么，在 TLP 传递的过程中到底经历哪些“磨难”呢？请看下图~

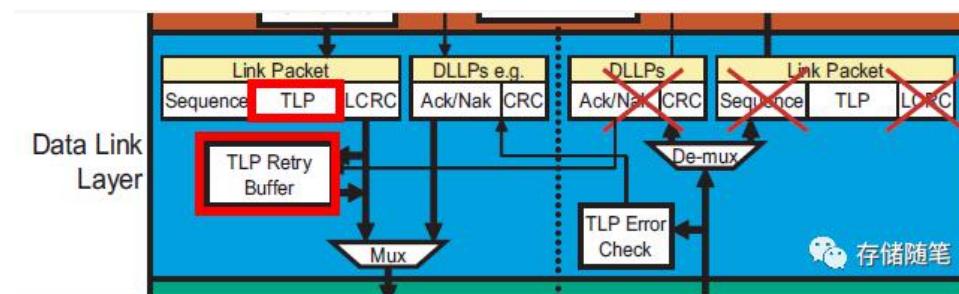


我们逐层解析一下这些“磨难”：

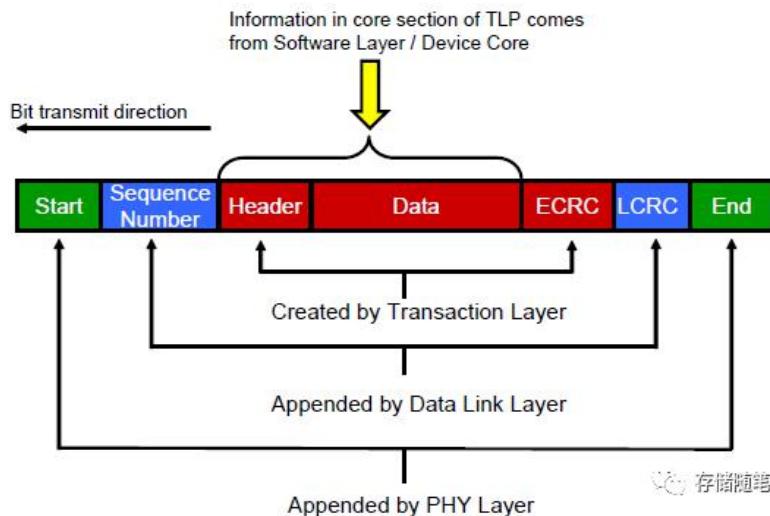
1. 发送端 Device Core 发送事务请求：数据读写，完成反馈(Completions)，信息(Message)等；
2. 事务层根据 Device Core 的请求，生成 TLP Header，加上 Device Core 提供的 data，最后加上 ECRC(End to End CRC)。此时 TLP 会放入事务层缓存(Virtual Channel Buffer)之中；
3. 当 TLP 传递至数据链路层时，会被加上 Squeeze Number 以及 LCRC(Link CRC)。此时，生成“加强版”TLP，并放入数据链路层的 Retry buffer；
4. 当 TLP 传至物理层时，被加上头和尾，到这里，TLP 在发送端就组装完毕咯；



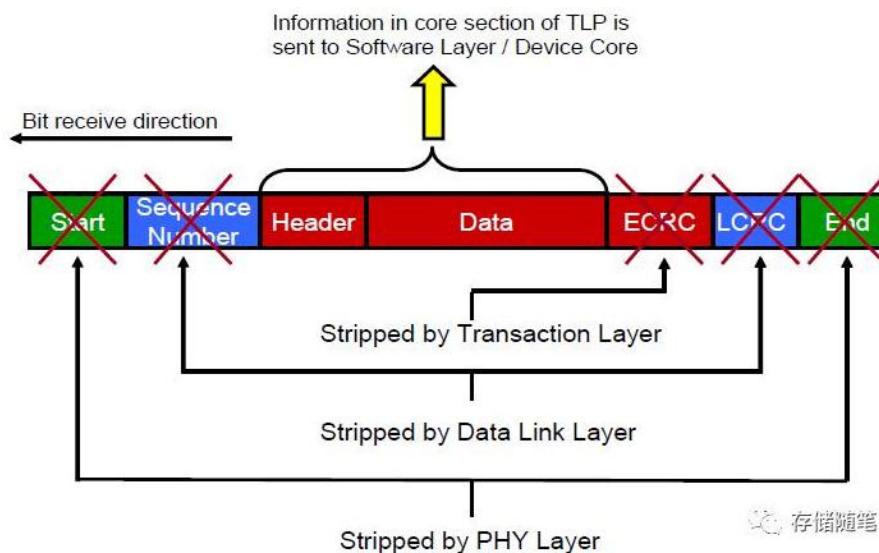
3. 当 TLP 传递至数据链路层时，会被加上 Squeeze Number 以及 LCRC(Link CRC)。此时，生成“加强版”TLP，并放入数据链路层的 Retry buffer；



4. 当 TLP 传至物理层时，被加上头和尾，到这里，TLP 在发送端就组装完毕咯；



5. 在接收端就跟发送端做的事情相反了，在物理层需要掐头去尾，然后传输至数据链路层；
6. 数据链路层收到传入的 TLP 后，通过计算 LCRC 验证传输是否正确，正确的话就去掉 Sequence Number 和 LCRC，将 TLP 传输至事务层；
7. 事务层接收到 TLP 之后，解析其内容，并将信息传给接收端 Device Core，至此，发送端传过来的组装 TLP 已拆解完毕。





2.2 TLP 事务处理方式

看过前面 TLP 的前世今生精彩大剧之后，想必大家应该都知道 TLP (Transaction Layer Packet) 在事务层的角色。如果不知道，就默默翻一下前面的文章哈~

TLP 很重要，也有很多种类。我们先来个全局的认识，看看 PCIe 到底定义了多少 TLP 种类呢？瞅下面的表格~

TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IOrd
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message Request with Data	MsgD
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk  存储随笔

这么多 TLP 啊，每个都是什么含义，到底是干什么的呢？不急，我们慢慢的揭开他们神秘的面纱~

之前的介绍中，我们提到过 PCIe 中总线事务有两大类：Non-Posted 和 Posted：

- (1) Non-Posted： 需要 completion 返回响应包；
- (2) Posted： 不需要 completion 返回响应包



Transaction Type	Non-Posted or Posted
Memory Read	Non-posted
Memory Write	Posted
Memory Read Lock	Non-posted
IO Read	Non-posted
IO Write	Non-posted
Configuration Read (Type 0 and 1)	Non-posted
Configuration Write (Type 0 and 1)	Non-posted
Message	Posted
AtomicOp	Non-Posted

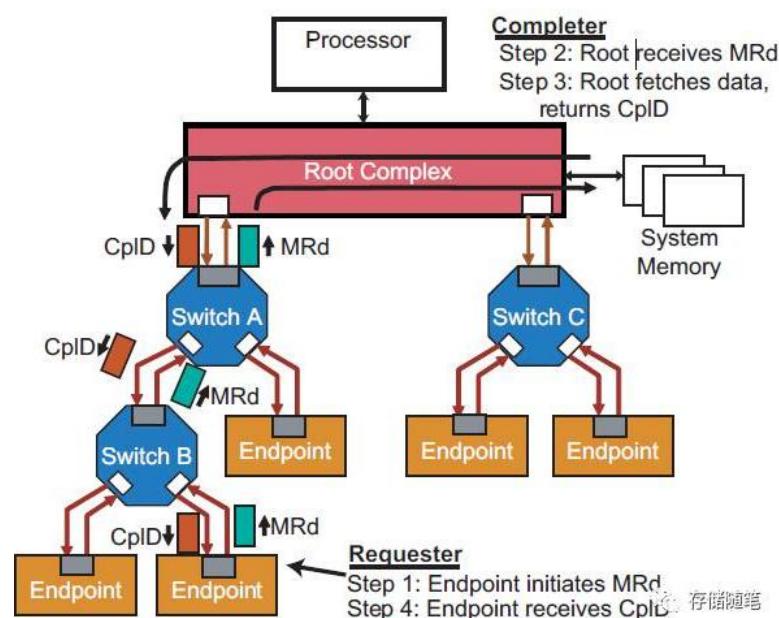
注：Memory Read Lock 只适用在兼容 PCI/PCI-X 的陈旧设备中，在这里就先忽略了。

我们先从 Non-Posted 和 Posted 阵营中的各挑选一些代表介绍一下 TLP 事务的具体作用。

Non-Posted Memory Read:

步骤：

- (1) PCIe 端初始化 Memory Read 请求；
- (2) 经过 Switch 到达 Root Complex；
- (3) Root 收到 Memory Read 之后就在系统缓存中抓取数据并回传完成报告 (Completion)；
- (4) 同样经过 Switch 到达设备端，设备端收到完成报告后结束此次事务请求。



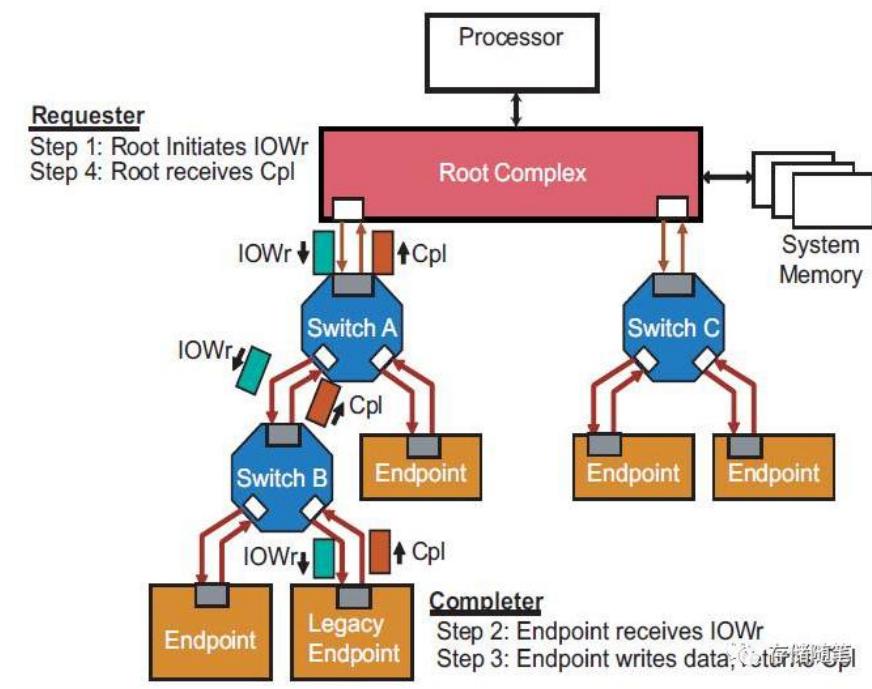


Non-Posted IO and Configuration Write:

步骤：

- (1) Root 初始化 IO Write 请求；
- (2) 请求和数据经过 Switch 到达设备端；
- (3) 设备端收到 IO Write 的请求以及数据之后，开始将数据写入到 PCIe 设备中，并回传完成报告(Completion)；
- (4) 同样经过 Switch 到达 Root，Root 收到完成报告后结束此次事务请求。

注意：Non-Posted 请求只能由处理器(也可以理解为 Root 端)发出！



Posted Memory Write:

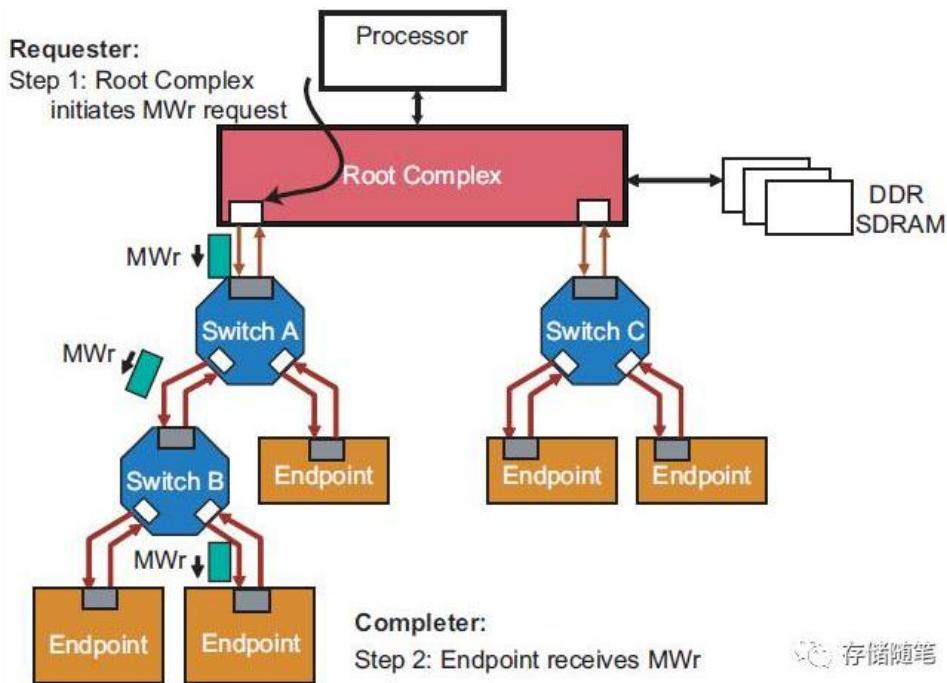
步骤：

- (1) Root 初始化 Memory Write 请求；

由于 Memory write 的请求是 Posted 方式，不要求有完成回报(Completion). 所以在 Memory write 请求发送之后，Root 不用等任何回应，就可以继续做其他事情啦~

- (2) PCIe 设备端收到 Memory Write 请求之后就开始专心执行写入动作，也不必准备完成回报(Completion).

不需要完成回报(Completion)虽然可以加速事务处理的速度，从而提高系统性能。但是，如果在设备端出现 Error，具体内容也不会回报给 Root。而是会给 Root 发送一条信息让其转告系统软件有 Error 需要处理。

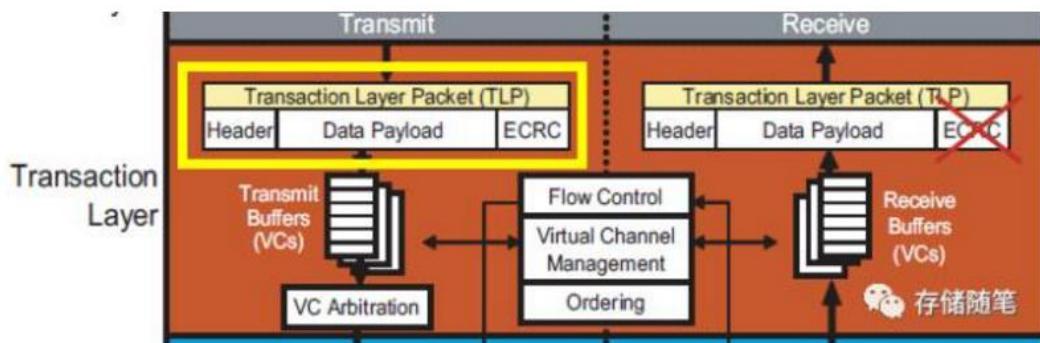




2.3 TLP 结构解析

经过之前文章的介绍，我们可以知道，PCIe 总线在事务层中利用 TLP (Transaction Layer Packet) 进行数据传输。那么，TLP 具体长什么样子呢？又有哪些关键内容？这个就是我们本文的重点。

TLP 主要包括三个部分：Header，Data Payload 和 ECRC(End to End CRC)。



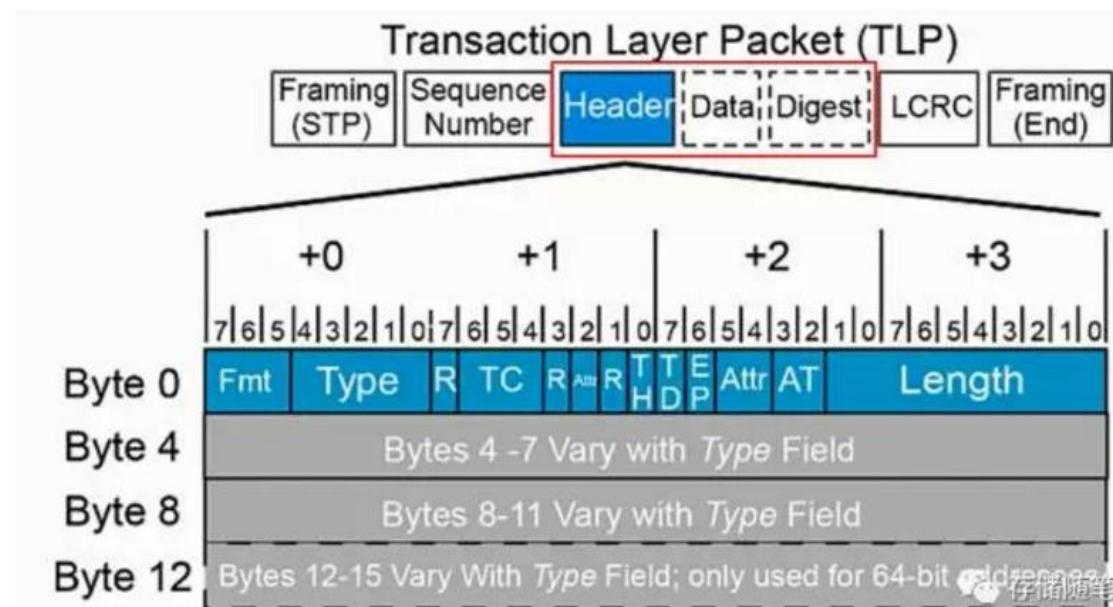
Header: 内容来自于 Device Core。大小一般为 3 或者 4 个 DWs (12~16bytes)。每个类型 TLP 中 Header 格式不尽相同，但是包含的信息大致相同。

Data: 内容也来自 Device Core。大小为 1~1024 DWs。

Digest/ECRC: 这部根据 Header 和 Data 内容计算得到。ECRC 可以选择性的附加。如果附加在 TLP 之后，大小一般为 1DW。

我们先介绍一下 Header 的结构和内容：

以一个通用的 4DWs TLP Header 为例，如下图。



接下来，我们针对上面 Header 的内容进一步解析：

Fmt[2:0] (Format): 3 bits, Byte0 Bit7:5, 具体含义如下：

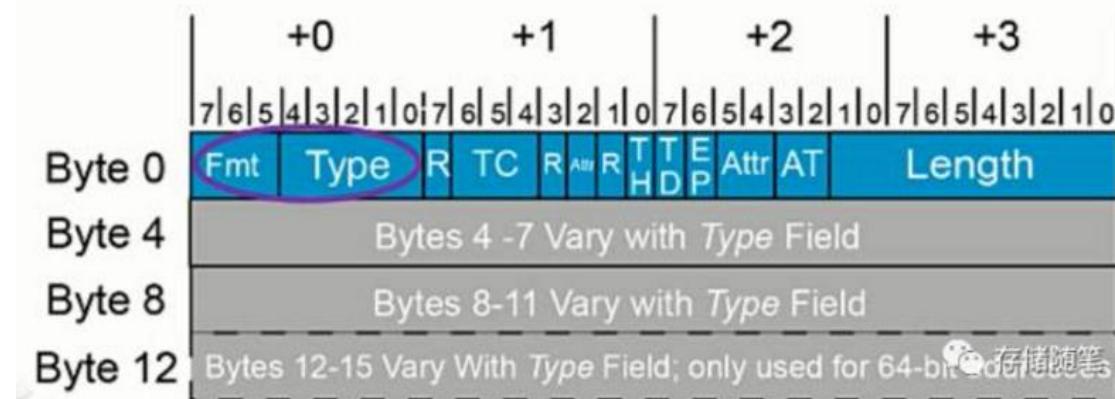


TLP Prefix Bit 7	Data Present Bit 6	Header Size Bit 5
0 = No TLP Prefix	0 = No Data (Read)	0 = 3 DWs
1 = TLP Prefix	1 = Data (Write)	1 = 4 DWs

Type[4:0]: 5 bits, Byte0 Bit4:0, 详细定义如下:

TLP Type	TYPE[4:0] Byte 0, Bit 4:0
Memory Request	00000
Memory Read Lock Request	00001
IO Request	00010
Configuration Type 0 Request	00100
Configuration Type 1 Request	00101
Message Request	10rrr
Completion	01010
Completion Lock	01011
Fetch and Add AtomicOp Request	01100
Unconditional Swap AtomicOp Request	01101
Compare and Swap AtomicOp Request	01110
Local TLP Prefix (Fmt[2:0] = 100)	0L ₃ L ₂ L ₁ L ₀
End-to-End TLP Prefix (Fmt[2:0] = 100)	1E ₃ E ₂ E ₁ E ₀

这里需要提一下，通常情况下 Type[4:0]与 Fmt[2:0]合起来定义具体的 TLP 事务类型，如下表：



举个例子：

Fmt[2:0]=001, 代表 Header 长度为 4DWs, 并且 no data 传输;

Type[4:0]=00000, 代表 Memory Request;

那么合起来就是, Memory Read Request(MRd).



TC[2:0](Trafic Class): 3bits, Byte1 Bit6:4, 总共定义了 8 个等级 Trafic Class.

000b = Traffic Class 0 (default)

001b = Traffic Class 1

⋮

}

Only Memory Read/Write Request and their associated Completion can use these TCs

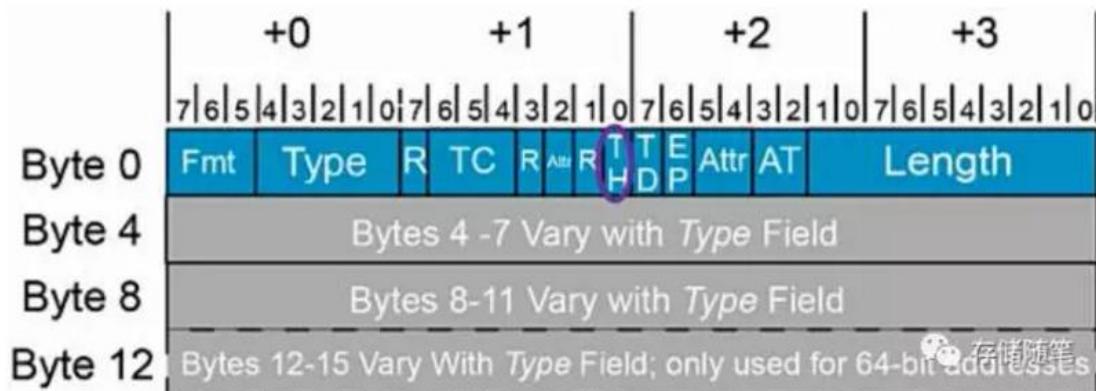
111b = Traffic Class 7

存储随笔

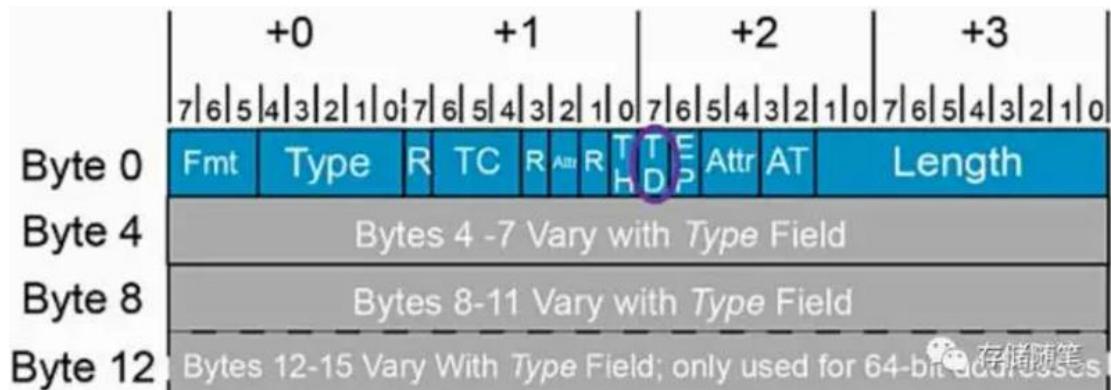
Traffic Class 顾名思义可以理解为交通等级。比如在生活中我们遇到救护车或者消防车时，由于救护车和救护车是跟时间赛跑，那么交通等级就高，一般的私家车(可以认为没有交通等级)就必须给救护车和消防车让行！可能这个类比不是很恰当，不过大致意思是这样的~

Attr[0](Attributes): 1 bit, Byte1 Bit2, 这个 bit 的功能就是是否对 TLP 进行排序(IDO, ID-based Ordering)。

TH[0](TLP Processing Hints): 1 bit, Byte1 Bit0, 如果 TH=1, 就代表在 TLP 中添加了 TLP 处理提示，可以告知系统以更有效的措施处理这个 TLP。



TD[0](TLP Digest): 1 bit, Byte2 Bit7, 如果 TD=1, 就代表 TLP 里面包含了 ECRC(End-to-End CRC)。





EP[0](Poisoned Data): 1 bit, Byte2 Bit6, 如果 EP=1, 就代表 TLP 传输的数据有误。

	+0	+1	+2	+3
	7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0			
Byte 0	Fmt	Type	R TC R _{AM} R _{HDP} TTE Attr AT	Length
Byte 4	Bytes 4 - 7 Vary with Type Field			
Byte 8	Bytes 8-11 Vary with Type Field			
Byte 12	Bytes 12-15 Vary With Type Field; only used for 64-bit addresses			

Attr[1:0](Attributes): 2 bits, Byte2 Bit5:4,

bit5=1 代表 enable PCI-X Relaxed Ordering 功能;

bit4=1 代表这个 TLP 没有主机缓存一致性的问题, 不需要缓存窥探措施;

扩展:

缓存一致性: 也就是说只有一块一级缓存, 所有处理器都必须共用它。在每一个指令周期, 只有一个幸运的 CPU 能通过一级缓存做内存操作, 运行它的指令。

窥探(Snoop): 窥探的思想是, 缓存不仅仅在做内存传输的时候才和总线打交道, 而是不停地在窥探总线上发生的数据交换, 跟踪其他缓存在做什么。所以当一个缓存代表它所属的处理器去读写内存时, 其他处理器都会得到通知, 它们以此来使自己的缓存保持同步。只要某个处理器一写内存, 其他处理器马上就知道这块内存它们自己的缓存中对应的段已经失效。

	+0	+1	+2	+3
	7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0			
Byte 0	Fmt	Type	R TC F _{AM} R _{HDP} TTE Attr AT	Length
Byte 4	Bytes 4 - 7 Vary with Type Field			
Byte 8	Bytes 8-11 Vary with Type Field			
Byte 12	Bytes 12-15 Vary With Type Field; only used for 64-bit addresses			

AT[1:0](Address Type): 2 bits, Byte2 Bit3:2, 这两 bits 针对 Memory 和 AtomicOp 请求声明存储地址类型的。



	+0	+1	+2	+3
Byte 0	Fmt	Type	R TC R Attr R HDP TTE Attr AT	Length
Byte 4	Bytes 4 - 7 Vary with Type Field			
Byte 8	Bytes 8-11 Vary with Type Field			
Byte 12	Bytes 12-15 Vary With Type Field; only used for 64-bit addresses			

Length[9:0]: TLP 传输的长度, 已 DW 为单位, 最大 1024 DWs.

	+0	+1	+2	+3
Byte 0	Fmt	Type	R TC R Attr R HDP TTE Attr AT	Length
Byte 4	Bytes 4 - 7 Vary with Type Field			
Byte 8	Bytes 8-11 Vary with Type Field			
Byte 12	Bytes 12-15 Vary With Type Field; only used for 64-bit addresses			

此外, 为了区分 Requester 与 Completer 之间的事务类型, PCIe Spec 定义了一个事务描述块(Transaction Descriptor Fields). 下图中 5 个红色区域合起来就称为一个事务描述块。

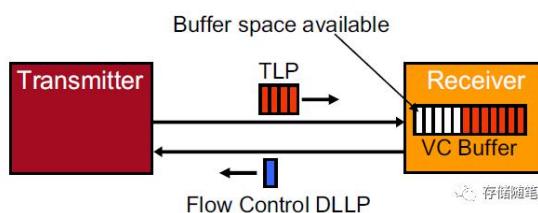
	+0	+1	+2	+3
Byte 0	Fmt	Type	R TC R Attr R HDP TTE Attr AT	Length
Byte 4	Completer ID			Cmpl Status B C M Byte Count
Byte 8	Requester ID		Tag	R L

Transaction ID 包含了 **Request ID** 和 **Tag**: 内容有 Request Device 具体的位置 (Bus/Device/Function)。主要是用于 Requester 发出事务的完成 TLP 回报。



2.4 Flow Control 机制概述

在 PCIe 协议中，如果要发送一个 TLP，就必须要保证接收端有足够的缓存(Buffer)来接收。为了实现这一功能，接收端会随时回报可用的缓存空间。



在接收端有一个缓存空间叫作 VC buffer，其中 VC 代表的是 Virtual Channel，翻译过来就是虚拟通道。VC buffer 可以存放从发送端传过来的 TLPs。

(1) PCIe 可以支持 8 虚拟通道(VC)，每个 VC 之间可以相互独立传输 TLPs，当一个 VC buffer 满的时候，并不影响其他 VC 继续传输。

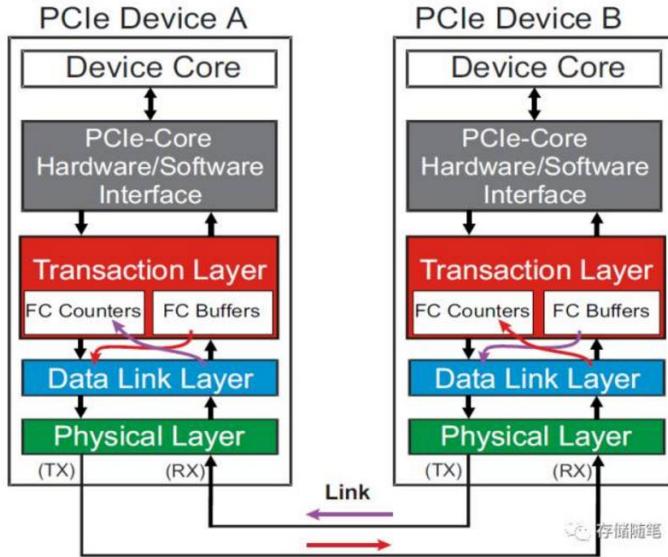
(2) 有关 VC 的详细内容会在后续章节介绍，本文就不再展开了~

在 flow control 中，还有一个很重要的概念不得不提一下：信用机制 (Credit-based Mechanism)。因为接受端的可用 VC buffer 是以信用机制的方式告知发送端，在这里，信用积分代表接收端 VC buffer 的可用空间。

接收端会通过发送 DLLPs (Data Link Layer Packets) 来告知发送端 VC buffer 的信用积分(也就是告知 VC buffer 可用空间)，当缓存空间快满的时候，发送端会停止发送 TLP，以防 VC buffer 容量不足而造成发送端传输的数据被丢弃，从而避免要求发送端重新发送刚才发送的数据，最终达到更加有效利用网络带宽。

其实，事务层(Transaction Layer)和链路层(Data Link Layer)是 Flow Control 机制的共同监护人。如下图：

Device A 和 Device B 均会把其事务层对应的可用空间(FC Buffer)告知链路层，链路层会在适当的时间生成 Flow control DLLPs 将可用空间的信息转送至接收端。（Device A to Device B, Device B to Device A）



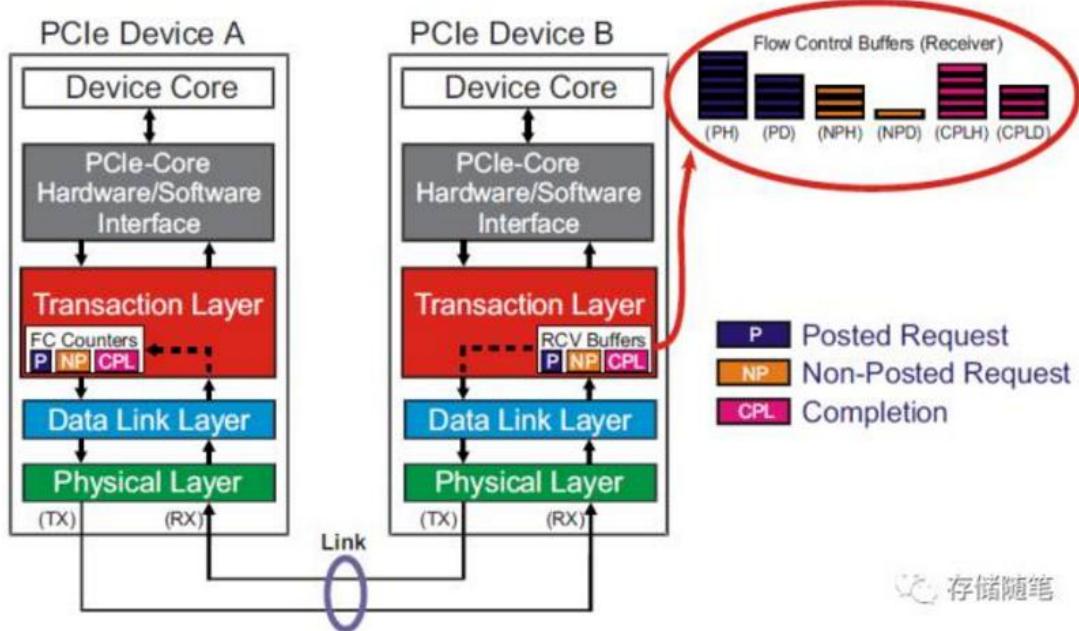
2.5 Flow Control 缓存架构及信用积分

在上一篇文章中，我们提到 PCIe 总线的 flow control（流量控制）是由事务层和数据链路层协调完成。发送端先以 TLP 的方式把数据发送至数据链路层，达到数据缓存之后会被分解为两个部分：Header 和 Data。

在 PCIe 总线中，VC buffer 总共由六个部分组成：

- ✧ PH(Posted Request Header): 存放 Memory Writes 和 Messages 请求的 TLP header;
- ✧ PD(Posted Request Data): 存放 Memory Writes 和 Messages 请求的 TLP Data;
- ✧ NPH(Non-Posted Request Header): 存放 Non-Posted 请求的 TLP header;
- ✧ NPD(Non-Posted Request Data): 存放 Non-Posted 请求的 TLP Data;
- ✧ CPLH(Completion Header): 存放 Read/Wirte Completions 请求的 TLP header;
- ✧ CPLD(Completion Data): 存放 Read/Wirte Completions 请求的 TLP Data;

注：有关 TLP 的结构与种类详见之前的文章。



存储随笔

所以，在发送 TLP 之前，发送端要确保接收端的 Header buffer 和 Data buffer 均有足够的空间接收数据。

通过阅读上一篇名为“Flow control 机制概述”的文章，相信大家应该大概了解了 Flow control 信用机制的含义。PCIe 协议中将接收端 VC buffer 的可用空间划分了很多单元，最小单元称为 Flow control 信用积分。

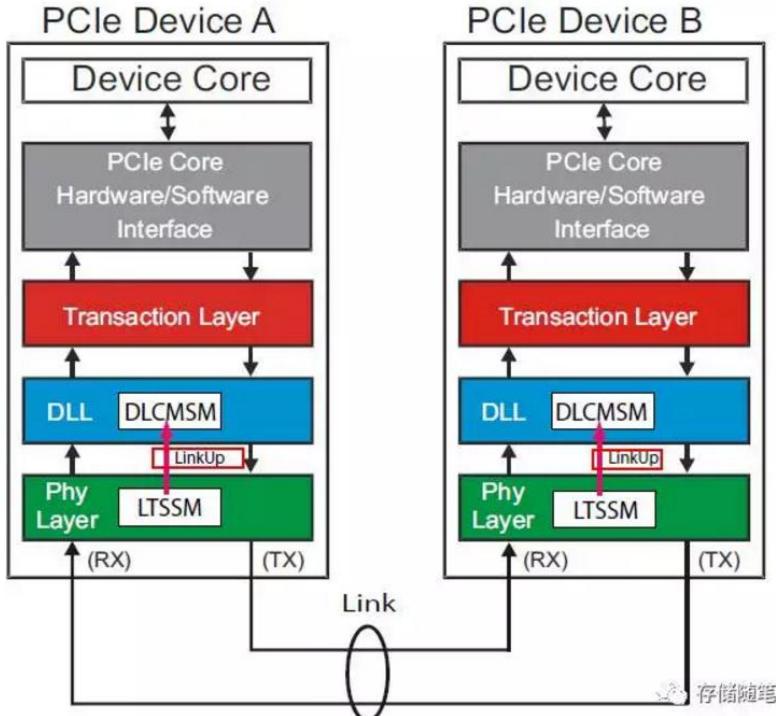
PCIe 总线对 Header 和 Data 的信用积分单元大小的定义是不一样的：

- ✧ **Header (for Competitions) Credit Unit: 4DWs;**
- ✧ **Header (for Requests) Credit Unit: 5DWs;**
- ✧ **Data (for Requests) Credit Unit: 4DWs;**

2.6 Flow Control 初始化

在 PCIe 总线中，任何事务传输之前，flow control 必须要初始化。如果 flow control 初始化未成功，那么任何 TLPs 都无法发送出去。

此外，Flow control 的初始化过程是在物理层 link training 完成之后进行，此时物理层中的 LinkUp 信号为触发状态，也即以为这物理层已经做好准备了。



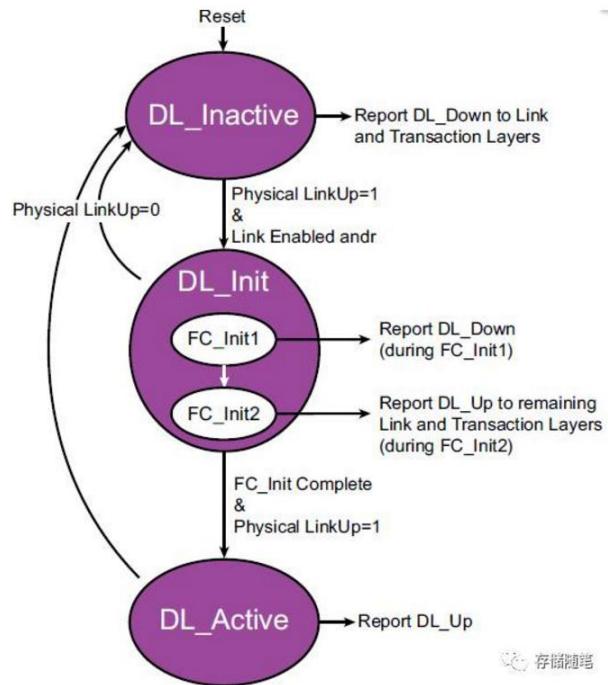
注: **DLCMSM**= Data Link Control and Management State Machine;

LTSSM= Link Training and Status State Machine;

对于所有的虚拟通道(VC0~7), Flow Control 初始化过程都是一样的, 默认状态是 enable VC0。

我们先来看一下 DLCMSM 状态图:

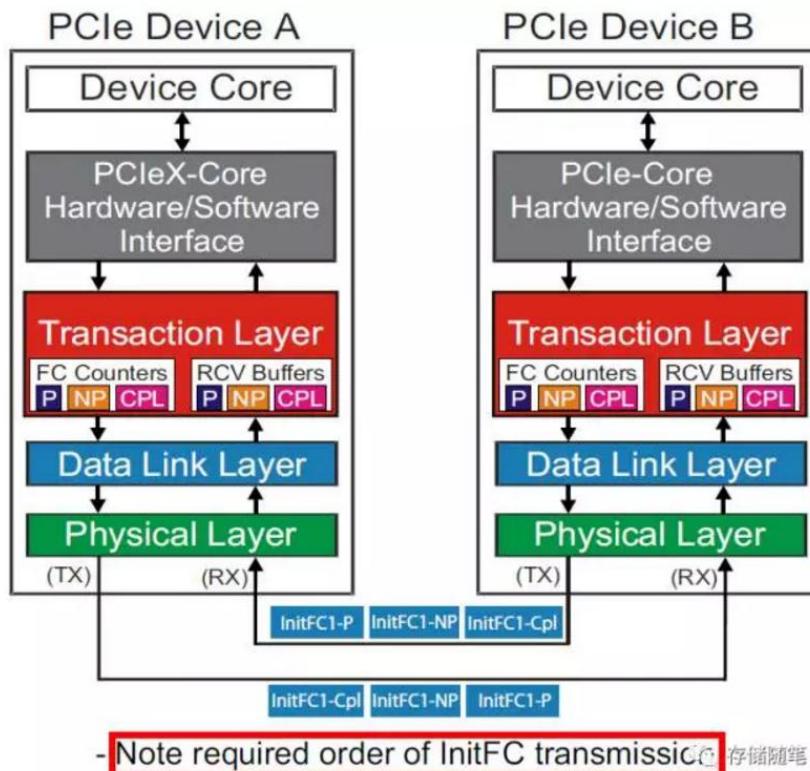
1. **DL_Inactive**: 物理层通知数据链路层当前 PCIe 链路不可用;
Reset 操作将 state machine 的状态调整为 **DL_Inactive**. 此时会向链路层以及事务层发送 **DL_Down** 信号通知此时状态;
2. **DL_Init**: 物理层正处于链路初始化状态;
当看到物理层传来的 **LinkUp** 信号(说明物理层做好准备了), state machine 的状态进入 **DL_init sub-state**: **FC_INIT1** 和 **FC_INIT2**. 这两个状态是 Flow control 的初始化的两个状态;
3. **DL_active**: 当前 PCIe 链路层处于正常工作状态;



接下来，我们针对 Flow control 初始化的两个阶段 **FC_INIT1** 和 **FC_INIT2** 作进一步的介绍。

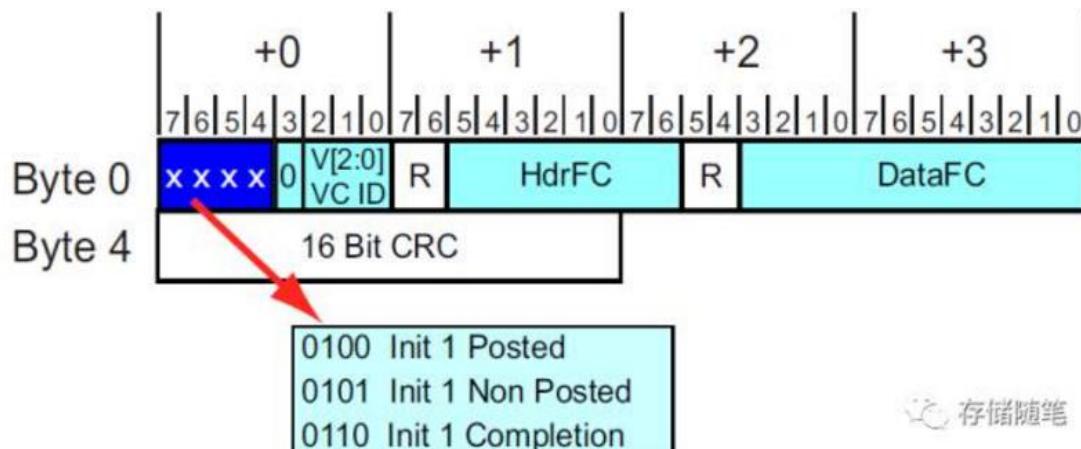
FC_INIT1:

当进入 FC_INIT1 阶段后，device 会持续依次发送 3 个 InitFC1 Flow Control DLLPs 初始化接收端的 VC buffer。





依照PCIe协议中的定义，InitFC1 Flow Control DLLPs包括Posted, Non-Posted, Completions三类，格式如下表：

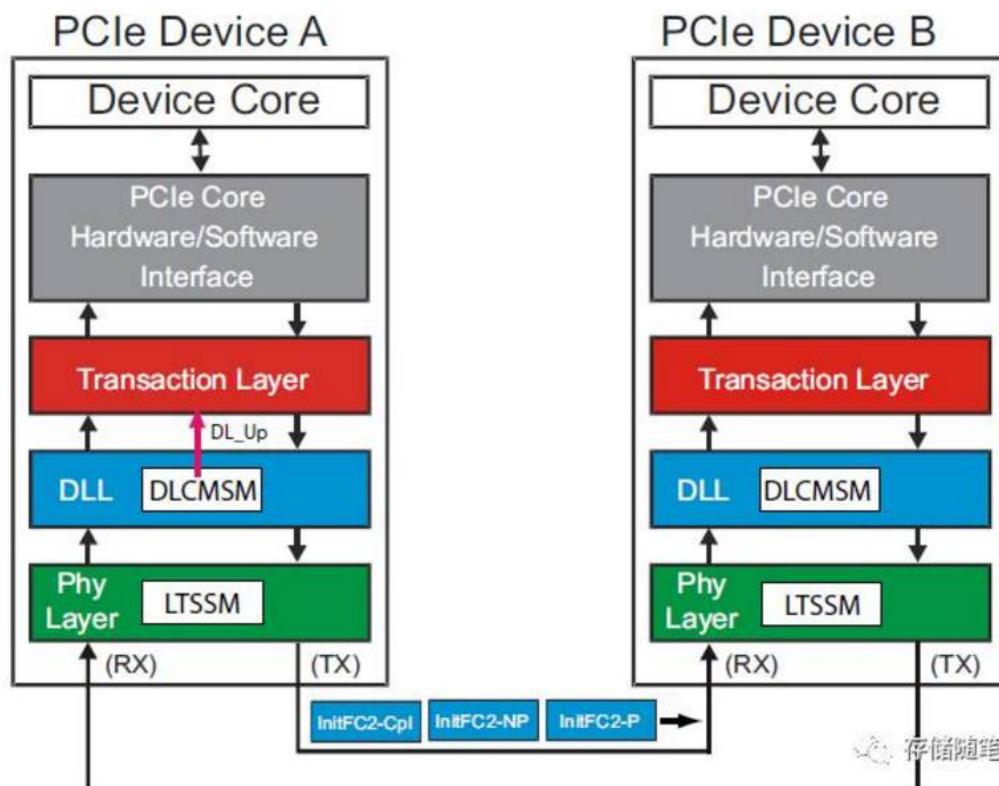


存儲隨筆

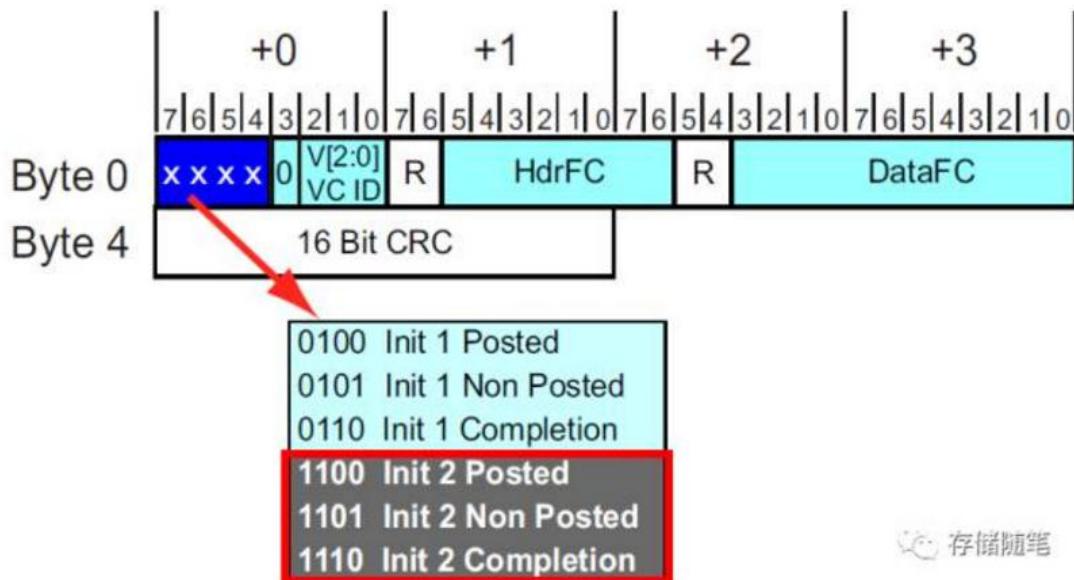
FC_INIT2:

FC_INIT1已经对Flow Control相关的缓存进行初始化，FCINIT2的作用主要是验证FC_INIT1的结果。FC_INIT2与FC_INIT1携带相同的Credit信息。

在FC_INIT2阶段时，Device会依次发送3个InitFC2 Flow Control DLLPs初始化接收端的VC buffer。成功发送完毕之后进入DL_active并回报DL_Up，告知事务层链路可以正常工作了。



与FC_INIT1一样，InitFC2 Flow Control DLLPs也包括Posted, Non-Posted, Completions三类，格式如下表：



存儲隨筆

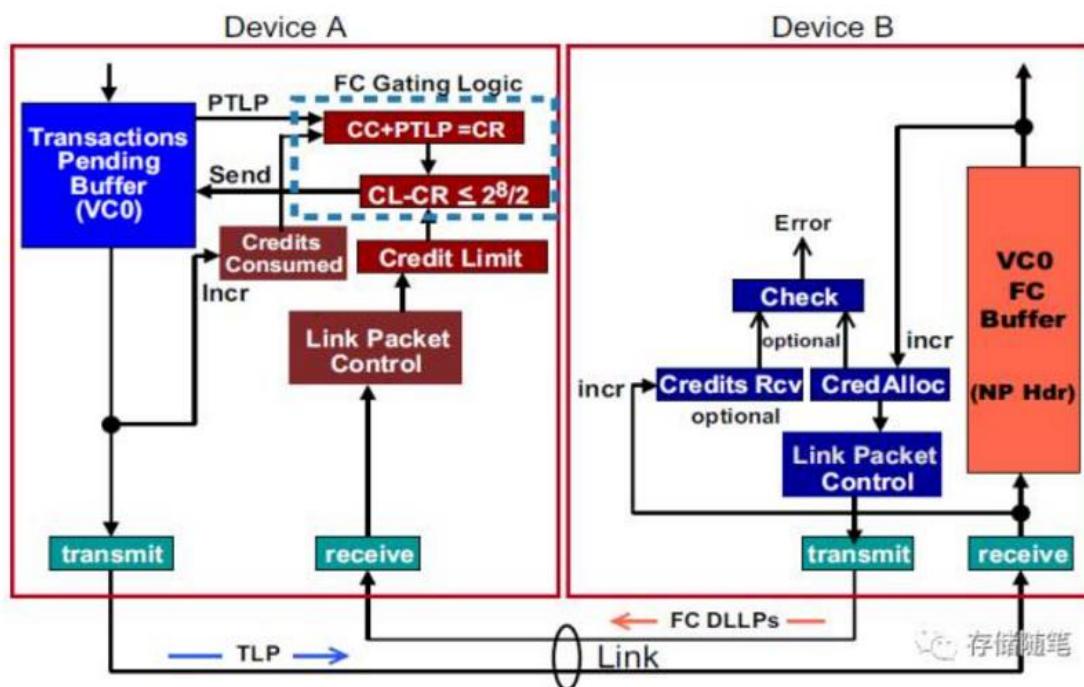
注：PCIe 总线上规定每隔 34us 发送一组 InitFC 报文.



2.7 Flow Control 的实现过程

前面针对 Flow control 的基本原理与组成进行了解析了，那么如何实现 flow control 这个功能呢？

要实现 flow control 功能，我们需要了解 Flow control 的控制逻辑单元。在之前的文章介绍中，我们了解了 VC buffer 有六个部分，每部分对应的 Flow 控制逻辑单元是一样的，所以，在这里，我们仅以 Non-Posted Request Header 对应的 Flow control 控制逻辑单元为例，如下图：



从上图中，我们可以看到 Flow control 控制逻辑单元主要包括：

发送端：

- ✧ **Transaction Pending Buffer:** 发送端用于存放正在等待在同一虚拟通道（VC）传输的 TLPs.
- ✧ **Credits Consumed Counter:** 在发送端在 VC buffer 中所有 TLPs 的信用单元总和，简称“CC”.
- ✧ **Credit Limit Counter:** 这部分由接收端的 VC buffer 的大小决定，简称“CL”。接收端会在 TLPs 传输的过程中不断发送 FC DLLPs 来更新 CL 的大小.
- ✧ **Flow Control Gating Logic:** 这个逻辑单元主要根据 CC, CL 以及待发送的 TLPs 的数量来判断接收端是否有足够的空间接收下一个 TLPs. 判断公式如下：

$$\{CL-(CC+PTLP)\} \bmod 2^{[Field\ Size]} \leq 2^{[Field\ Size]/2}$$

注释: Header Field Size=8, Data Field Size=12. PTLP= Pending TLP, 待发送 TLP.

接收端：

- ✧ **Flow Control Buffer:** 存储传进来的 Data 和 Header 数据.
- ✧ **Credits Allocated:** 接收端 VC buffer 已经分配的存储空间记录.
- ✧ **Credit Received Counter:** 这部分主要记录接收端接收的所有 TLPs 对应的信用单位总和。

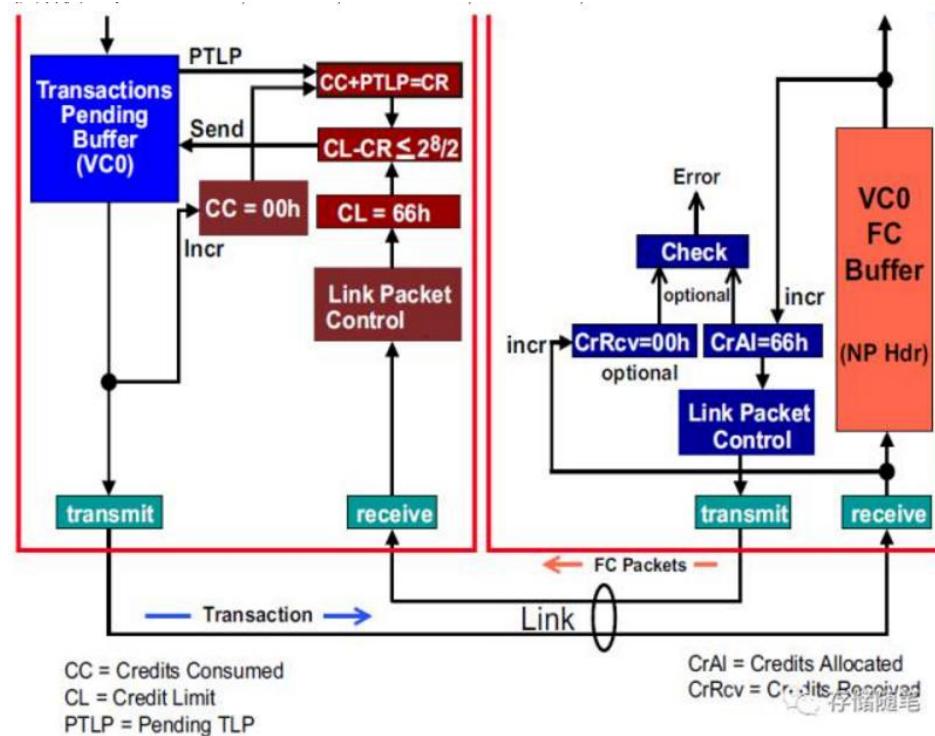


我们看一个例子看一下 Flow control 具体过程：

1. 初始状态下 Flow control 控制逻辑单元：

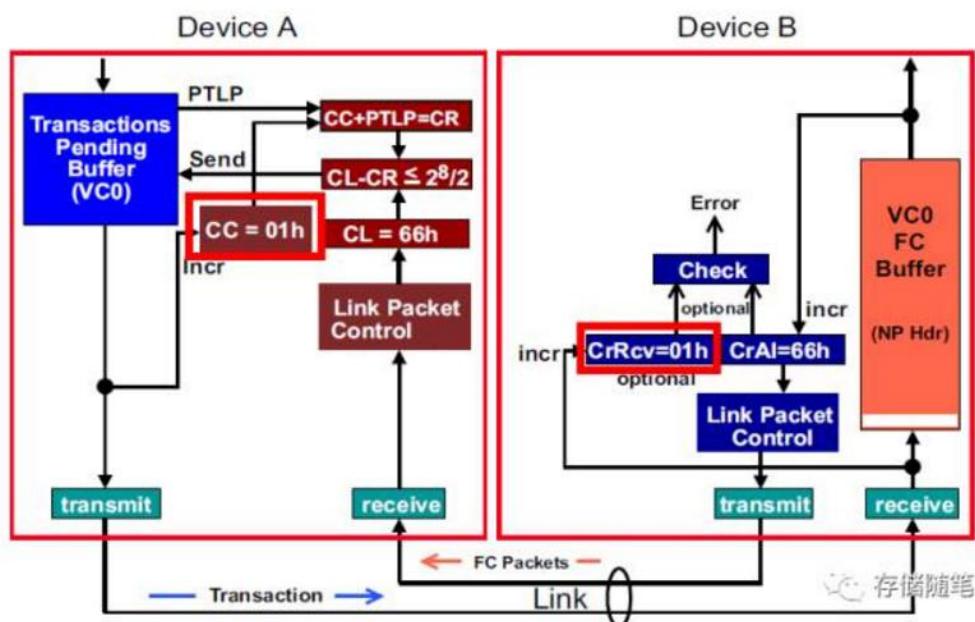
在这里例子中，我们假设 VC buffer 大小为 2KB，之前的文章介绍过 Non-Posted Header 的信用大小为 5DW,也就是 20Bytes, 所以 Flow control 单元数目最大为 $2\text{KB}/20\text{Bytes}=102\text{d}=66\text{h}$.

初始状态下：CC=00h, CL=66h, CrRcv=00h, CrAl=66h,



2. 传输一个 TLP 之后的 Flow Control 控制逻辑单元：

此时，CC=01h, CrRcv=01h,



3. VC buffer出于满的状态时的 Flow Control 控制逻辑单元:

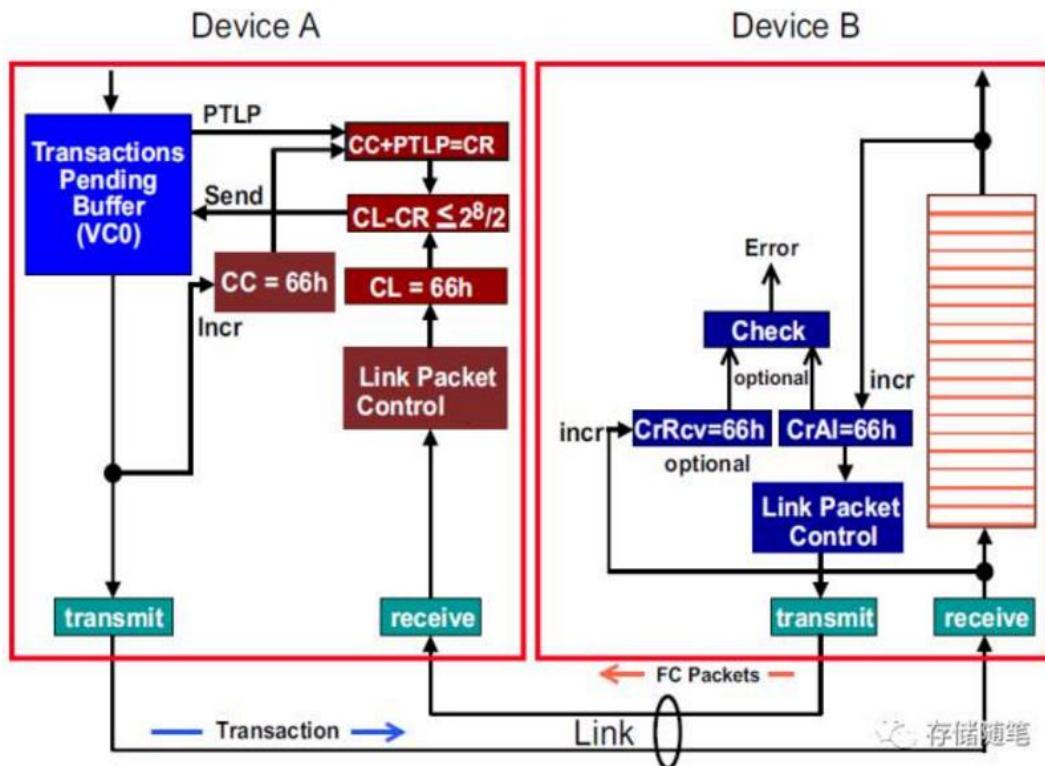


发送端: CC=CL=66h, CR=CC+PTLP=66h+01h=67h,

CL-CR=66h-67h=66h+99h=FFh 不满足判断公式.

接收端: CrRcv=CrAl=66h,

此时, 说明了 VC buffer 已经满了, 暂停传输 TLPs.

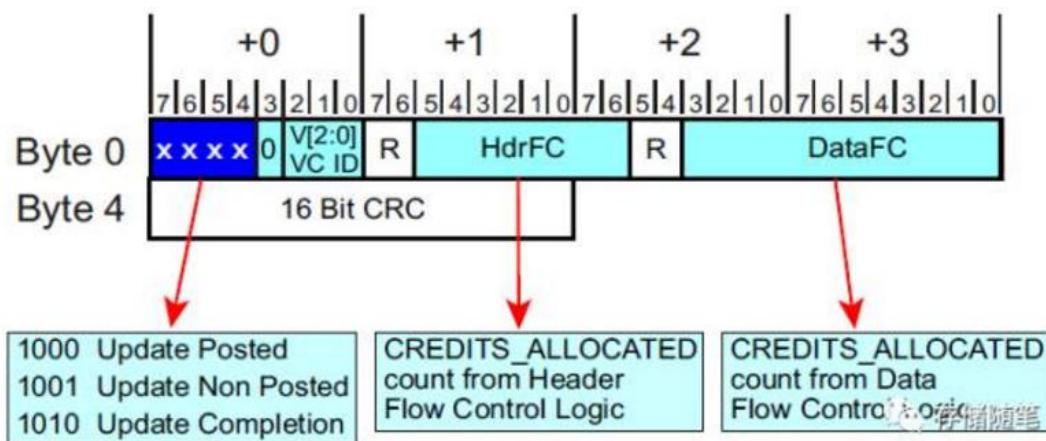


4. VC buffer 清空部分数据的 Flow Control 控制逻辑单元:

接收端 VC buffer 清空了 3 个 Non-Posted Header 数据, 此时 VC buffer 有了新的可用空间, 可以继续接受 TLPs. 但此时发送端并不知道, 怎么办呢?

放大招！！！

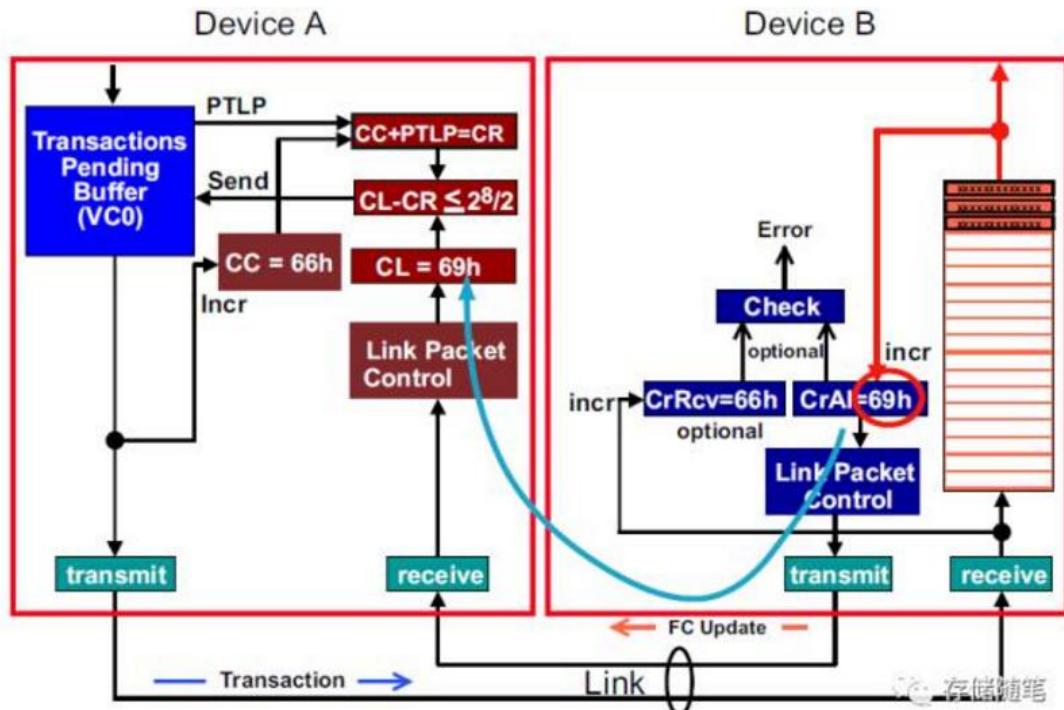
PCIe 协议中有定义一个 Flow Control update packets, 也即 Flow Control update DLLPs. 格式如下图,



有了大招就好办了, 接收端通过发送 Flow control update DLLPs 告知发送端: “我这边腾出地方了, 可以传 TLPs 过来咯! ”



同时，CL 更新为 69h, CC=66h, CR=66h+01h=67h,
CL-CR=69h-67h=02h<128, 满足判断公式，TLPs 可以继续传输。





2.8 事务排序机制

与其他协议一样，当同一个通信等级(Traffic Class, TC)的多个事务(Transactions)同时通过同一个通道时，PCIe对这些事务设置了一些排序(Transaction Ordering)规则。

这样做好处有以下几点：

- ✧ 保持与传统总线的兼容性。比如PCI, PCI-X等；
- ✧ 保证事务的完成具有准确性，并且按照设计人员的意愿完成；
- ✧ 避免死锁的状况发生；
- ✧ **最大限度的优化PCIe总线的传输效率。**

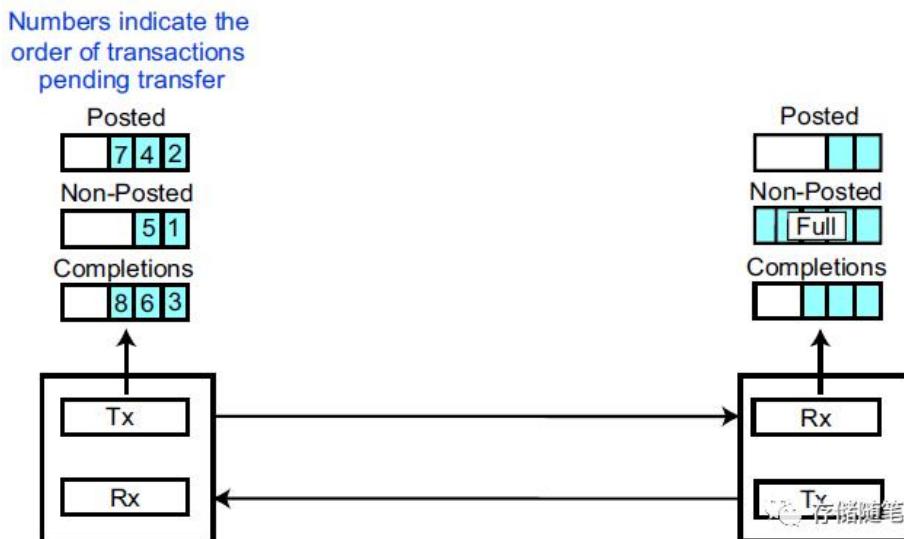
注释：死锁是指两个以上的设备在访问临界资源时，相互等待对方释放这些资源，而无法访问这些资源的情况。

在PCI总线上，只支持**Strong Ordering**传送规则，而在PCIe总线中新增了**Relaxed Ordering(RO)**和**ID-based Ordering(IDO)**传递方式。

Strong Ordering:

何为 Strong Ordering? 顾名思义，Strong，就是很强壮，很彪悍的意思。Strong Ordering 强制总线上的TLP按照先来后到的方式进行传递，一视同仁，不管是否有特殊情况，均不允许插队和绿色通道。但是事情总有轻重缓急，对于一些紧急的状况，这个强制规则难免会影响到总线的传递效率和用户体验。

我们看下面一个例子，



- ✧ 所有的TLPs类型统一编号；
- ✧ 发送端VC buffer有8个待传输的TLPs，依次编号为#1~8，这8个TLPs分为三类：
- ✧ **Posted:** 2, 4, 7;
- ✧ **Non-Posted:** 1, 5;
- ✧ **Completions:** 3, 6, 8;



- ◆ 接收端对应的 VC buffer 如上图，其中，Non-Posted buffer 已满，而 Posted、Completions buffer 则仍有可用空间；

此时，接收端 Non-Posted buffer 已满，TLP 1,5 则需要暂停发送。由于这 8 个 TLPs 属于同一 VC，需要按照 Strong-Ordering 规则排序。所以 TLP 2,3,4 对应接收端 VC buffer 即使有可用空间，也必须等 TLP 1 传输完成之后才能发送。

这不就是浪费时间嘛，浪费别人的时间，就等于谋杀呀~Strong-Ordering 太不厚道咯~~~

Relaxed Ordering(RO):

相比 PCI, PCIe 则深得中庸之道，更加善解人意。PCIe 支持的 Relaxed Ordering 的传递规则不会要求 TLPs 严格遵守先来后到，也意味着根据轻重缓急找到最佳的方案，提高数据的传输效率。

不过，Relaxed Ordering 是有条件的。因为每个 TLP 只能对应一个 TC(通信等级)，而这个 TC 有只能对应唯一一个 VC(虚拟通道)。所以说：

- ◆ VC 相同的 TLPs 遵循 Relaxed Ordering 规则，
- ◆ VC 不同的 TLPs 则没有事务排序的要求，不必遵循 Relaxed Ordering 规则。

VC 相同的 TLPs 需要遵循的排序规则具体如下表：

Can Row pass Column? (Col 1)		Posted request (Col 2)	Non-Posted Requests		Completion (Col 5)
Non-Posted Requests	Read Request (Row B)		Read Request (Col 3)	NPR w/ data (Col 4)	
	Posted request (Row A)	a) No (unless b) applies) b) Y/N (if RO=1 or IDO=1 and different ReqID)	Yes (deadlock avoidance)	Yes (deadlock avoidance)	a) Y/N (unless b) applies) b) Yes (special case for PCI/PCI-X bridges)
	Read Request (Row B)	a) No (unless b) applies) b) Y/N (if IDO=1 and different ReqID)	Y/N	Y/N	Y/N
	NPR w/ data (Row C)	a) No (unless b) applies) b) Y/N (if RO=1 or IDO=1 and different ReqID)	Y/N	Y/N	Y/N
	Completion (Row D)	a) No (unless b) applies) b) Y/N (if RO=1 or IDO=1 and different CmplID / ReqID or if completion for IO or Config Write)	Yes (deadlock avoidance)	Yes (deadlock avoidance)	a) Y/N (for completions from different read requests) b) No (for completions from same read request)

先解释一下表格中的 Yes, Y/N, No 对应的含义：

Yes: 代表第二个事务必须在第一个事务之前通过，也就是强制性"超车"；

Y/N: 没有排序要求；

No: 代表第二个事务绝对不允许在第一个事务之前通过，也就是"超车"违法。

我们解析一下表格中的 A2b, B2, C2b, D2b 的内容来阐述一下 Relaxed-Ordering 的效果：

A2b: 当 RO=1，也即 Enable Relaxed-Ordering 规则。此时，Memory Writes or Messages 被允许超车之前的 Memory Writes or Messages 而抢先通过；

B2: 无论 RO bit 是否被置起来，Read Requests 均不被允许超车之前的 Memory Writes or Messages 而抢先通过；



C2b: 当 RO=1, IO writes 和 Configuration writes 被允许超车之前的 Memory Writes or Messages 而抢先通过；

D2b: 当 RO=1, Competitions 被允许超车之前的 Memory Writes or Messages 而抢先通过；

ID-based Ordering(IDO):

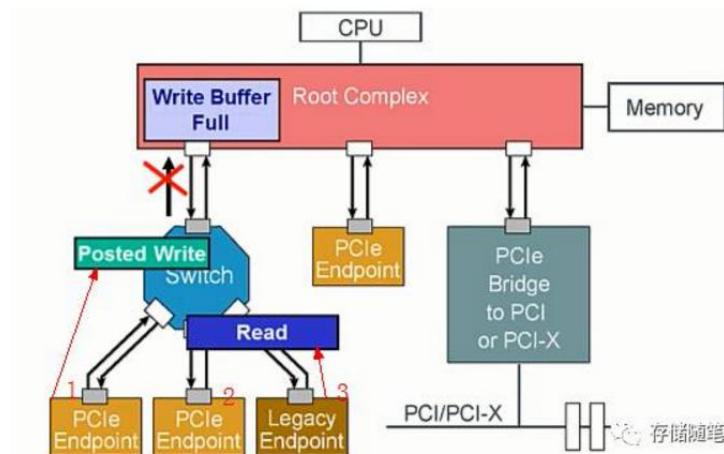
IDO 的模型是在 PCIe V2.1 版本之后新增的功能。该模型引入了"数据流"（Stream）的概念，即：

- ✧ 相同数据源发送的 TLPs 属于同一数据流；
- ✧ 不同数据源发送的 TLPs 属于不同的数据流；

IDO 模型允许不同数据流的 TLPs 之间不必遵循事务排序的约定。

我们来看个例子：

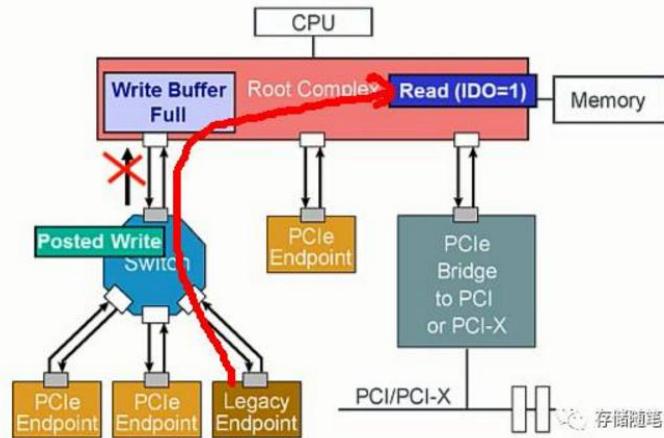
- ✧ 设备 1 发送了 Posted Write Request, 但是, 此时 Write Buffer 已处于满的状态, 所以 Posted-Write 被锁定在 Switch。
- ✧ 随后, 设备 3 发送了 Non-Posted Read Request, 通过 Switch 时, 正好撞到了上面表格 B2 所指规则(**B2:** 无论 RO bit 是否被置起来, Read Requests 均不被允许超车之前的 Memory Writes or Messages 而抢先通过;), 所以, Read request 同样被锁定在 Switch。



此时，就要靠 IDO 这位大神出手咯~

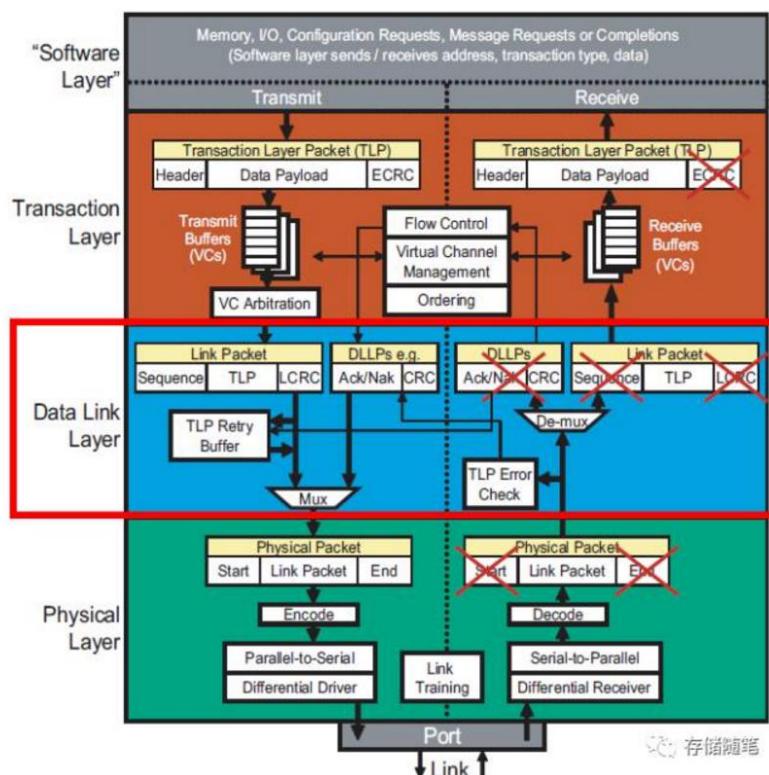
依照 IDO 模型的思想，Posted Write request 和 Read request 的数据源不一样，可以不遵循事务排序规则。

所以，当 PCIe 总线 enable IDO 功能时，Read request 可以很开心的通过 Switch，这个时候，Posted Write 只有羡慕嫉妒恨的份儿咯~~~



3.0 数据链路层概述

之前的文章中，我们提到“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、数据链路层和事务层，并最终到达 Device Core。”



从上图中，我们也可以明显的看到，Data Link Layer 在 PCIe 总线中处于承上启下的作用，保证来自事务层的 TLPs 在 PCIe 总线中正常的传递。

- ◆ 在发送端，当 Transaction Layer 事务层的 TLPs 传进来时，Data Link Layer 会在 TLP 前后两端分别加上 Sequence 以及 LCRC 部分。



◆ 在接收端，数据链路层接收到 TLP 报文之后进行拆解，剥离 Sequence 和 LCRC 部分，再传送至事务层。

此外，数据链路层使用了 Retry 与监控机制(Ack/Nak)来保证数据传输的一致性和完整性。

来自事务层的 TLPs，再加上前缀 Sequence 和后缀 LCRC 之后，会首先暂存在数据链路层的 TLP Retry Buffer，然后再发送至接收端。发送端会根据接收端传到的 Ack/Nak DLLP 决定是否需要重新发送 TLP。如果不需要重新发送，则将其从 TLP Retry Buffer 里清除。

注意: DLLP 不同于 TLP, 是产生与数据链路层, 中止与数据链路层。DLLP 并不是有 TLP 加上 Sequence 和 LCRC 组成的，具有单独的格式。

3.1 数据链路层 DLLP 结构及类型

上篇文章数据链路层概述中提到”DLLP 不同于 TLP，是产生与数据链路层，中止与数据链路层。DLLP 并不是有 TLP 加上 Sequence 和 LCRC 组成的，具有单独的格式”。我们本文就主要介绍一下 DLLP 的结构与类型。

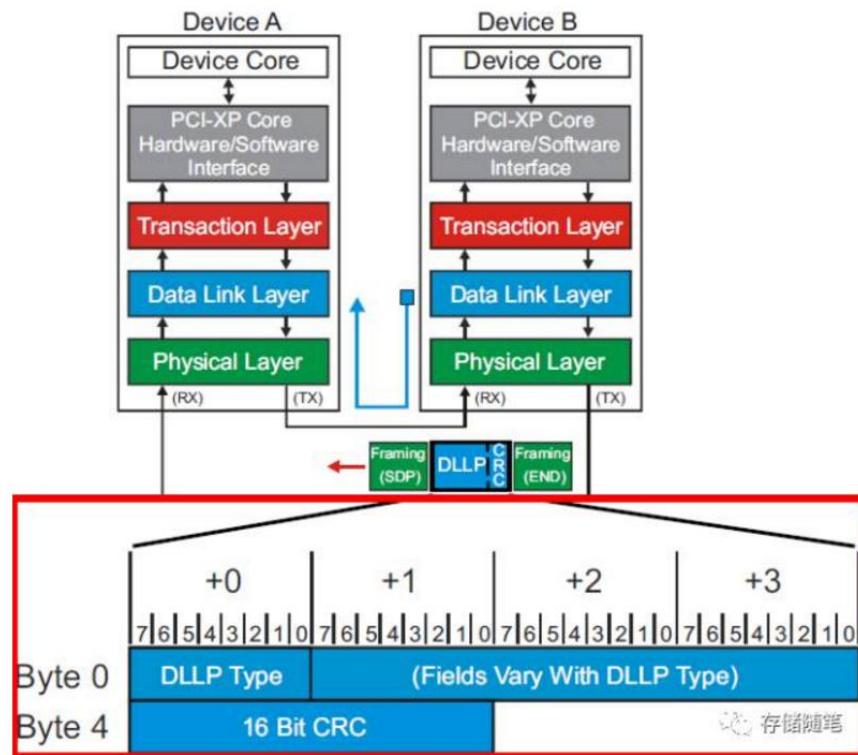
先看一张图，在数据链路层中 TLP 和 DLLP 的样式如下：



在数据链路层中，TLP 要加上前缀 Sequence ID 和后缀 LCRC，如上图。而 DLLP 在数据链路层产生，包含的信息有 DLLP Type，Attitude Field 以及 CRC。

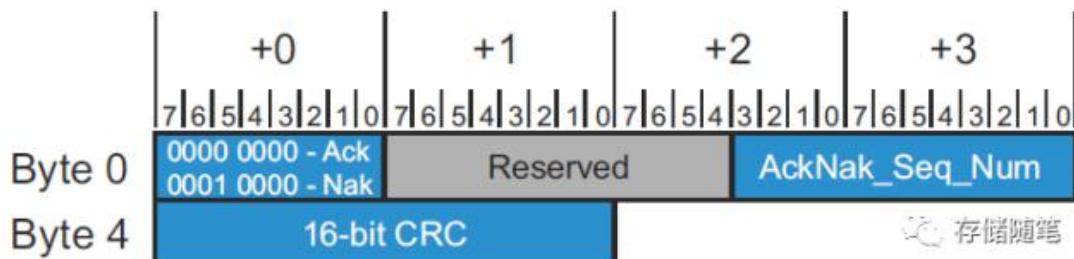
需要指出的是，DLLP 里面的 CRC 为 16 位，与 TLP 的后缀 LCRC(32 位) 不同，不要混淆咯~

一个 DLLP 的大小是固定的，为 6 Bytes，如下图。第一个 Byte 为 DLLP Type，第二到四个 Bytes 是于 DLLP 类型相关的属性参数，最后两个 Bytes 是 CRC 区域。如下图。

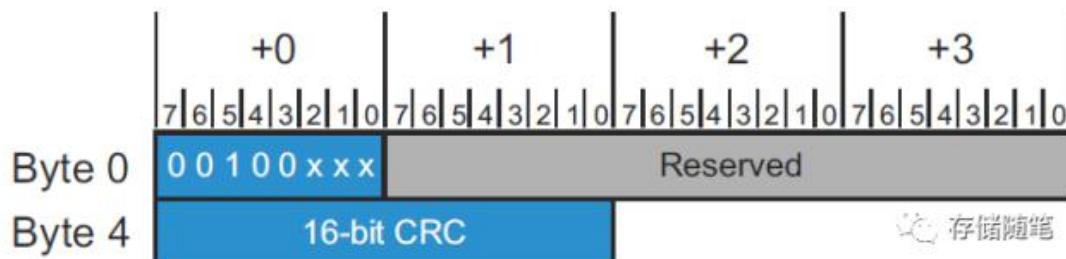


DLLP 的类型大致分为四类: Ack/Nak, Power Management, Flow control 以及用户自定义。

Ack/Nak DLLP 格式:



Power Management DLLP 格式:



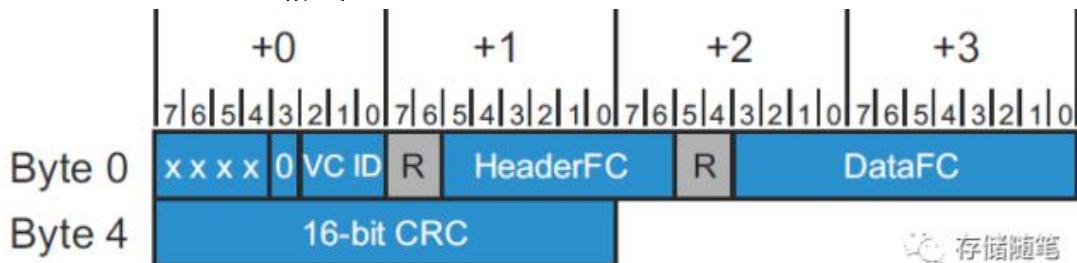
- ✧ 0010 0000b = PM_Enter_L1
- ✧ 0010 0001b = PM_Enter_L23
- ✧ 0010 0011b = PM_Active_State_Request_L1



◆ 0010 0100b = PM_Request_Ack

Ps: 电源管理部分的详细内容在后续章节介绍。

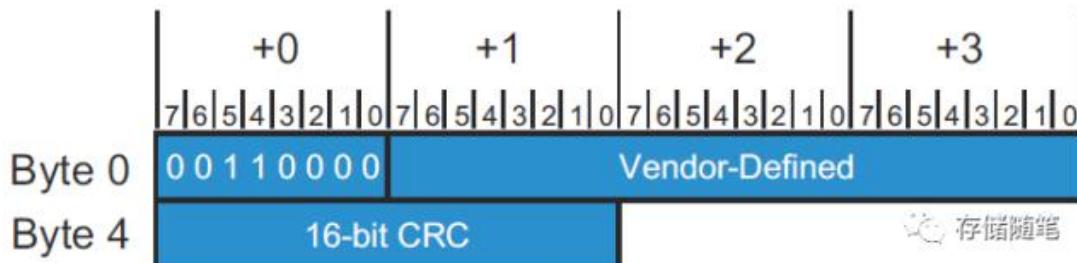
Flow Control DLLP 格式:



存储随笔

Ps: 这部分其实在前面的章节介绍了，具体请翻阅前面的文章。

用户自定义 DLLP 格式:



存储随笔

附录：DLLP 详细列表



DLLP Type	Type Field Encoding	Purpose
Ack (TLP Acknowledge)	0000 0000b	TLP transmission integrity
Nak (TLP Negative Acknowledge)	0001 0000b	TLP transmission integrity
PM_Enter_L1	0010 0000b	Power Management
PM_Enter_L23	0010 0001b	Power Management
PM_Active_State_Request_L1	0010 0011b	Power Management
PM_Request_Ack	0010 0100b	Power Management
Vendor Specific	0011 0000b	Vendor Defined
InitFC1-P	0100 0xxxb	TLP Flow Control (xxx = VC number)
InitFC1-NP	0101 0xxxb	TLP Flow Control
InitFC1-Cpl	0110 0xxxb	TLP Flow Control
InitFC2-P	1100 0xxxb	TLP Flow Control
InitFC2-NP	1101 0xxxb	TLP Flow Control
InitFC2-Cpl	1110 0xxxb	TLP Flow Control
UpdateFC-P	1000 0xxxb	TLP Flow Control
UpdateFC-NP	1001 0xxxb	TLP Flow Control
UpdateFC-Cpl	1010 0xxxb	TLP Flow Control
Reserved	Others	Reserved

存储随笔



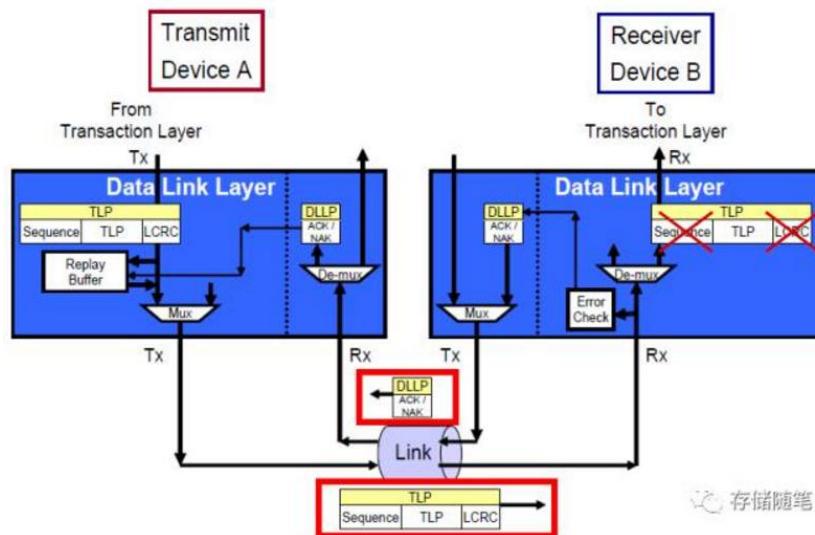
3.2 数据链路层 Ack/Nak 机制

在上一篇文章“DLLP 结构与类型”中，我们有说到，数据链路层会产生多个 DLLP。其中有两个 DLLP 分别是 Ack DLLP 和 Nak DLLP。这两个 DLLP 均是由接收端传至发送端，也可以理解为一种反馈机制。

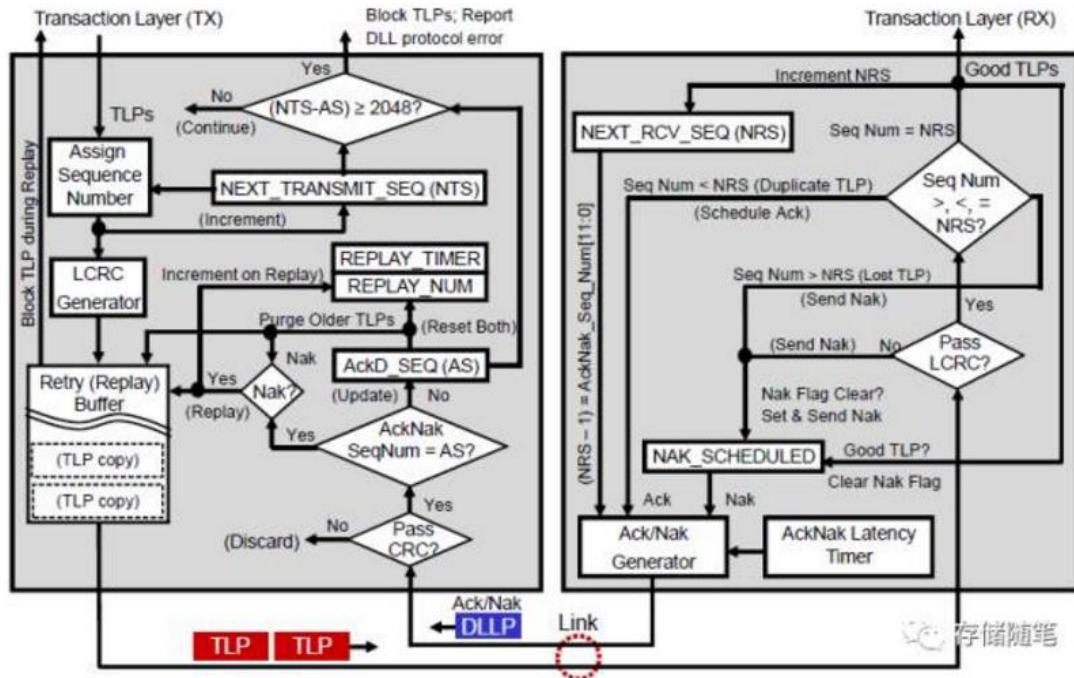
- ◆ **Ack DLLP**: 表示接收端收到了来自发送端的正确的 TLP 报文；
- ◆ **Nak DLLP**: 表示接收端有来自发送端的 TLP 报文没有被正确接收，需要发送端重新发送。

我们通过下面这张图，先来大致了解一下 Ack/Nak 的工作机制：

1. 发送端数据链路层传送一个 TLP(Sequence+TLP+LCRC)，通过 Link，到达接收端。
2. 接收端接收到来自发送端数据链路层的 TLP 报文之后，先检验 LCRC，在检验 Sequence ID。
3. 当 Sequence ID 和 LCRC 检验均正确时，接收端返回 Ack DLLP 告知发送端：“您发送的 TLP，我方已正确接收，请知悉！”。
4. 当 Sequence ID 或者 LCRC 检验中，发现哪怕一个错误，接收端都会返回 Nak DLLP 告知发送端：“对不起，您发送的 TLP 没有被正确接收，请您再发送一下”。



现在大概知道 Ack/Nak 的工作机制了吧？继续往下看，再来一张图。



是不是有点晕？小编第一次看到的时候也晕~

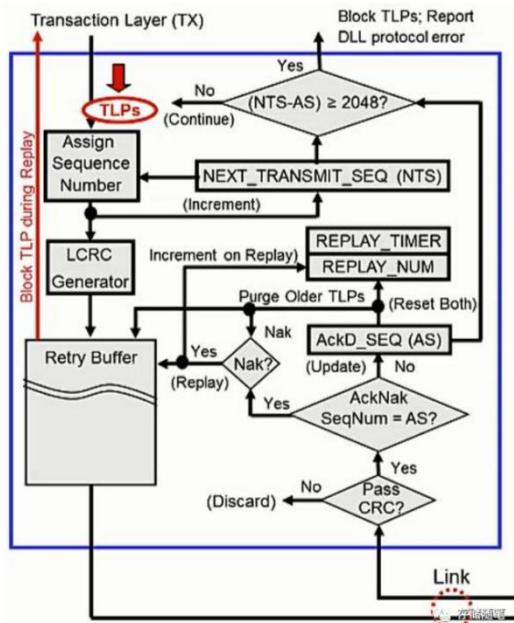
上面这幅图是数据链路层 Ack/Nak 机制详细的结构图，理解了上面这幅图就基本掌握了 Ack/Nak 机制。

不要被上面这幅图吓到，我们下面就一步一步地解析：

1. 从发送端事务层传送的 TLP 达到数据链路层

如下图红色圈内所示。

不过，这里需要注意的是：当 Retry buffer 是满的或者正在执行重新发送 TLP 的状态，数据链路层将会锁定 TLP 传送，不再接收。

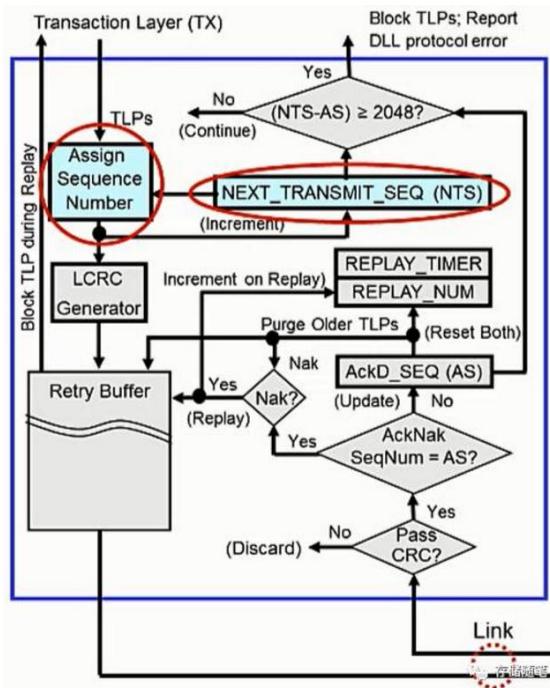




2. 为 TLP 分配 Sequence ID

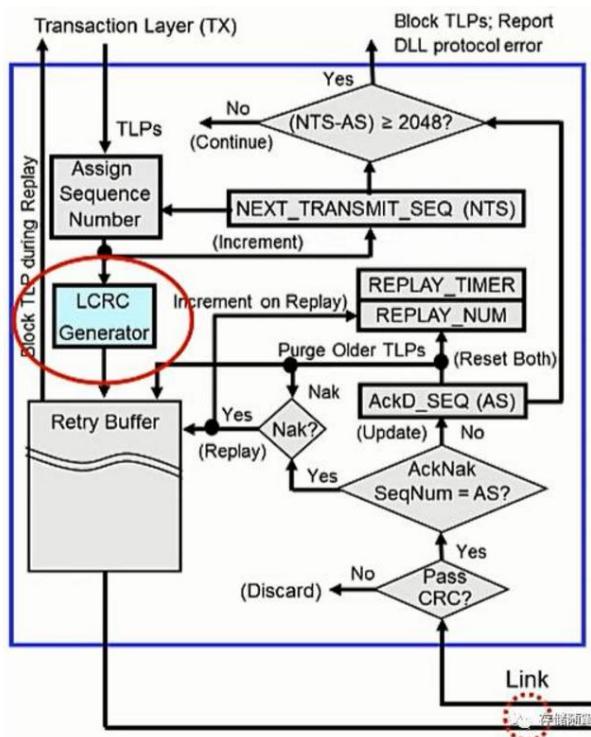
当 TLP 到达数据链路层之后，第一件事就是被分配 Sequence ID。也可以理解为给 TLP 一个身份编号，以便于后续的检验工作。

这里还要提一个参数: `NEXT_TRANSMIT_SEQ`, 简称 NTS, 是一个 12 位的 Sequence ID 计数器, 初始值为 0, 最大取值为 4095。一个 TLP 被分配 Sequence ID 之后, NTS 会加 1, 然后把累加后的数值再赋值给下一个 TLP。



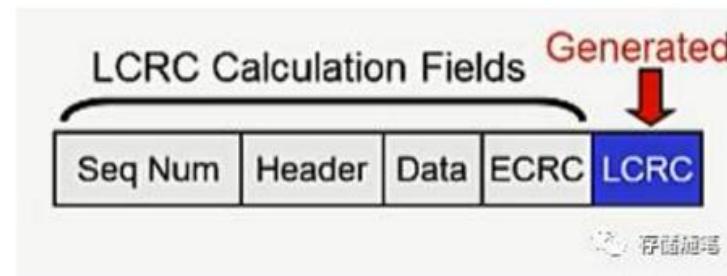
3. 为 TLP 增加 LCRC

TLP 分配 Sequence ID 之后，下一步就是生成 LCRC。





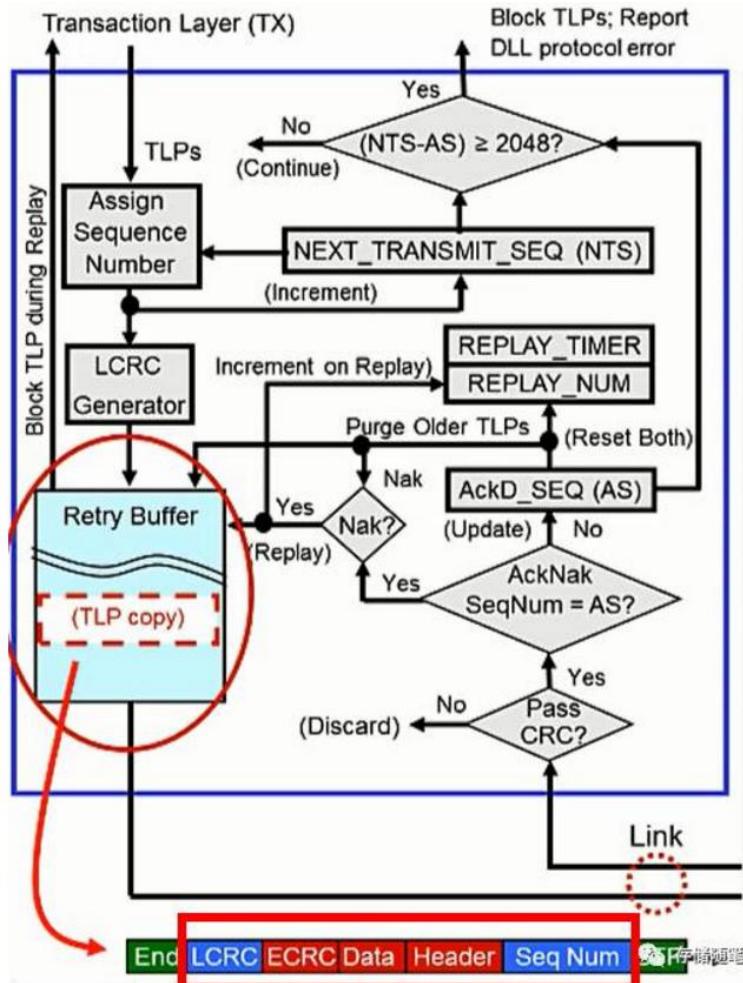
LCRC 为 32 位，基于事务层传送的 TLP 和数据链路层分配的 Sequence ID 生成。



4. 将 TLP 在 Retry Buffer 备份

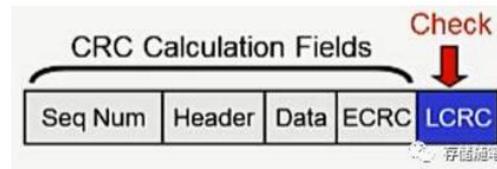
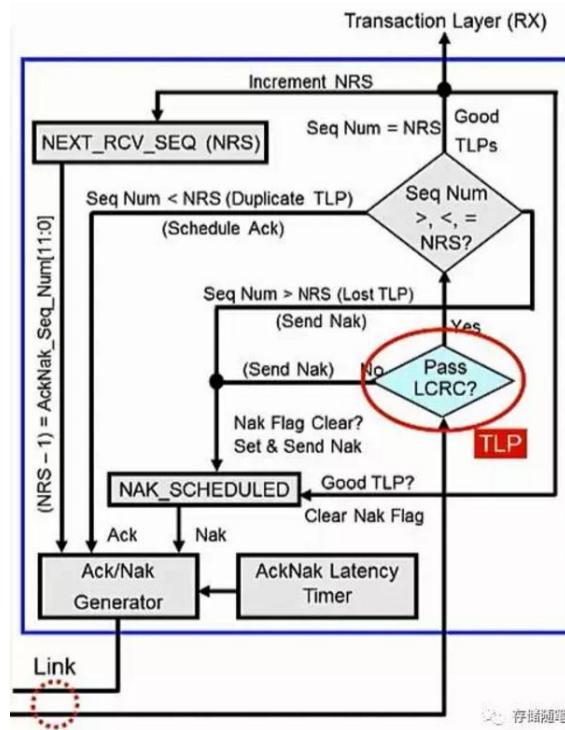
TLP 在加上前缀 Sequence ID 和后缀 LCRC 之后，会在 Retry Buffer 里面完整备份。

- ✧ 单个 TLP 最大占用的 Retry Buffer 大小为：4122 Bytes (2 bytes Sequence ID + 16 bytes Header + 4096 bytes Data + 4 bytes ECRC + 4 bytes LCRC).
- ✧ PCIe Spec 中并没有规定 Retry Buffer 大小，不同的设计采用的大小不同，但是必须保证在 TLP 传输的过程中不能遇到瓶颈。



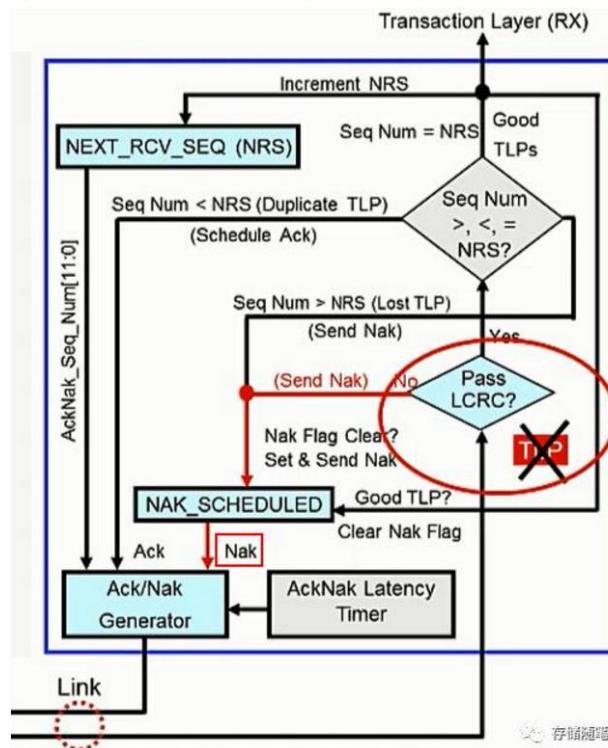
5. 接收端对接收的 TLP 进行 LCRC 检查

接收端接收到发送端传来的 TLP 后会先根据 Sequence ID, Header, Data, ECRC 计算 LCRC，然后再跟传进来的 LCRC 对比，检查是否一致。



6. LCRC 檢查 fail

當 LCRC 檢查 fail 時，會舍棄剛才傳進來的 TLP，並將 NAK_SCHEDULED 标志位置起來，給發送端回報 NAK DLLP。此外將期望接收到





7. LCRC 檢查 OK，檢查 Sequence ID。

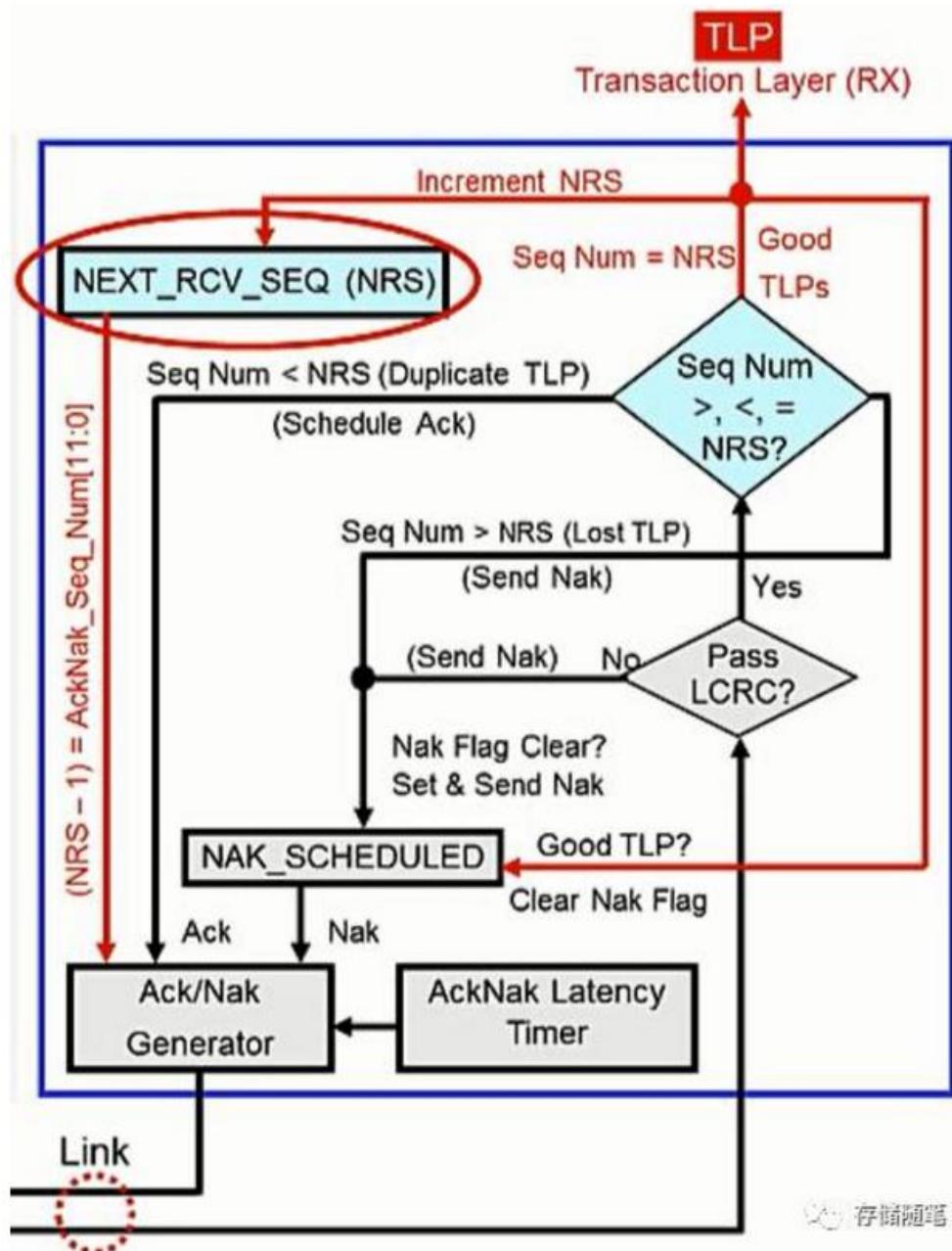
當 TLP 的 LCRC 檢查 OK 之後，接收端繼續檢查 Sequence ID。這裡出現了新的參數：NEXT_RSV_SEQ，簡稱 NRS，用於追蹤下一個期望獲得的 TLP 的 Sequence ID，NRS 有 12 位，取值範圍 $0^{\sim}4095$ 。

檢查 Sequence ID 時分為三個情況：

- ✧ Sequence ID=NRS，代表是 TLP 接收正確；
- ✧ Sequence ID<NRS，代表是 TLP 是重複的；
- ✧ Sequence ID>NRS，代表是 TLP 有發生丟失的情況；

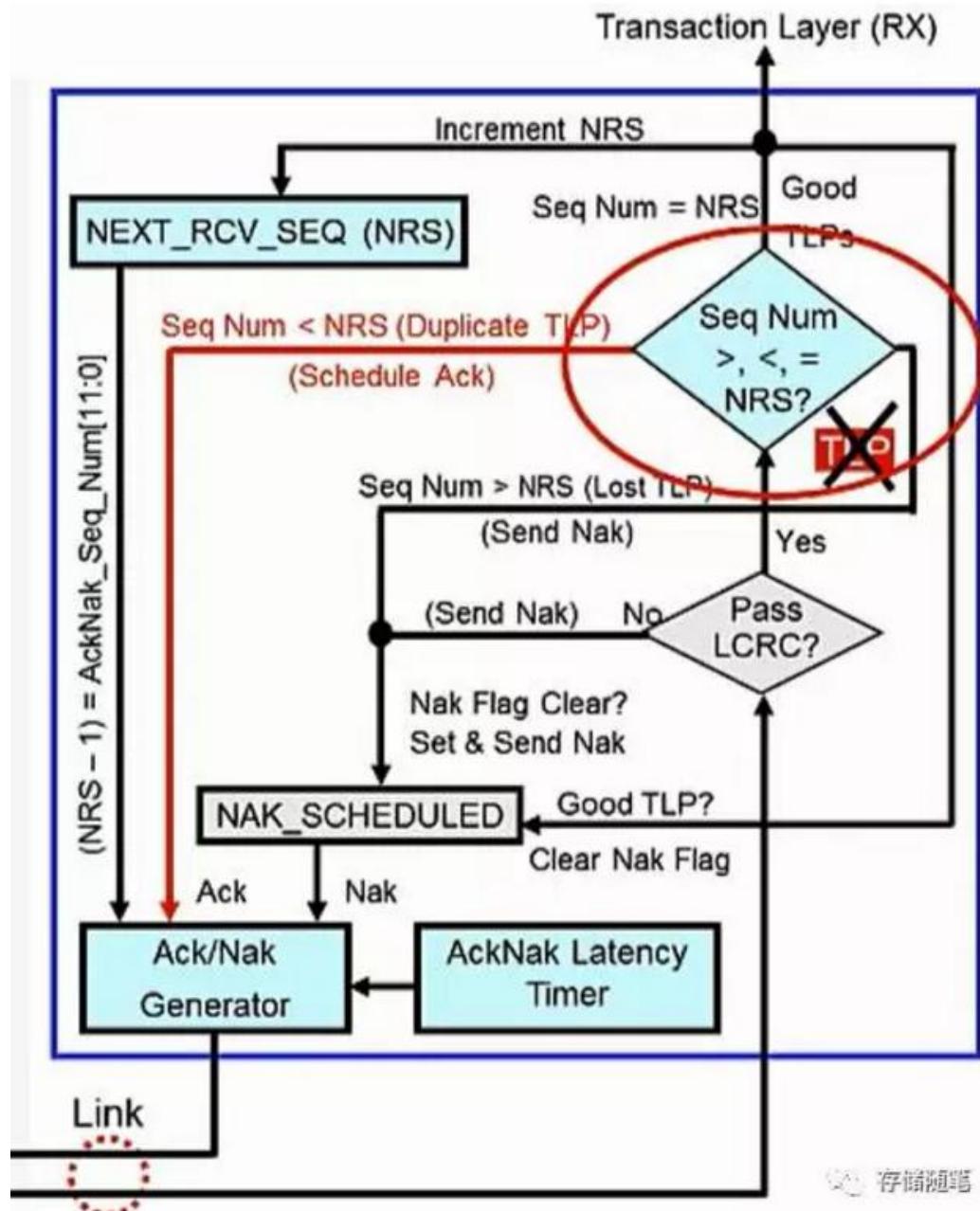
(1) **當 Sequence ID=NRS 時，**

這個情況下，正確接收 TLP，並將 TLP 傳送至上次事務層。同時 NRS 要加 1，準備接收下一個 TLP。另外還要給發送端回報 Ack DLLP 告知發送端已正確接收 TLP。



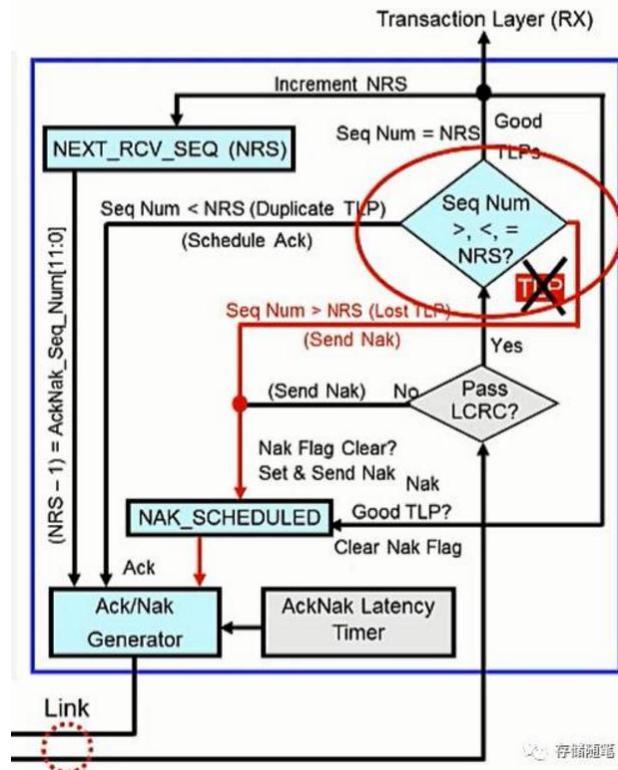
(2) 当 Sequence ID<NRS 时，

这个情况下，代表接收端收到了重复的 TLP。当下收到的这个 TLP 会被舍弃，此时 NRS 保持原有数值，并给发送端回报上一个有效 TLP 的 Ack DLLP。



(3) 当 Sequence ID>NRS 时，

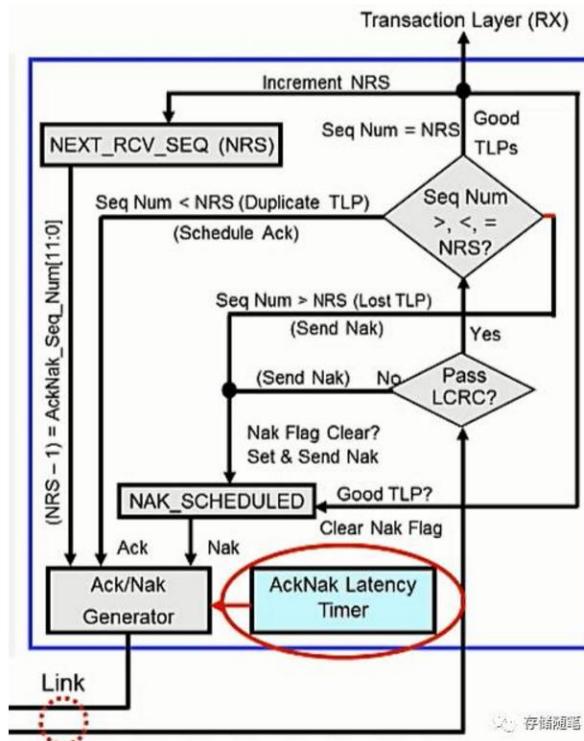
这个情况下，代表当下 Sequence ID 之前的 TLP 丢失。当下收到的这个 TLP 会被舍弃，此时 NRS 保持原有数值，NAK_SCHEDULED 标志位被置起，并给发送端回报上一个有效 TLP 的 Ack DLLP。



8. Ack/Nak Latency Timer

接收端还有一个重要的参数：Ack/Nak Latency Timer。延迟时间不是固定的，与 Link Width, Max Payload 有关。

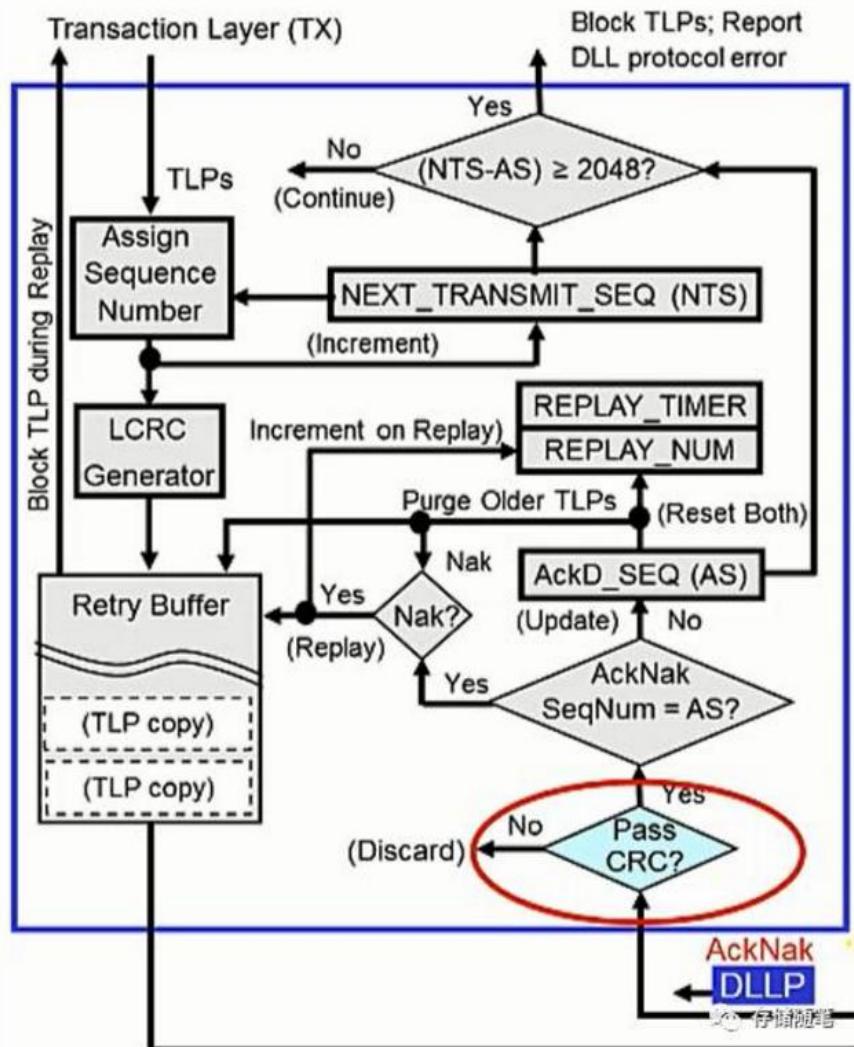
Latency timer 超时，Ack/Nak 生成器会给发送端发送 Ack DLLP。发送 Ack DLLP 之后，Latency Timer 会重置。



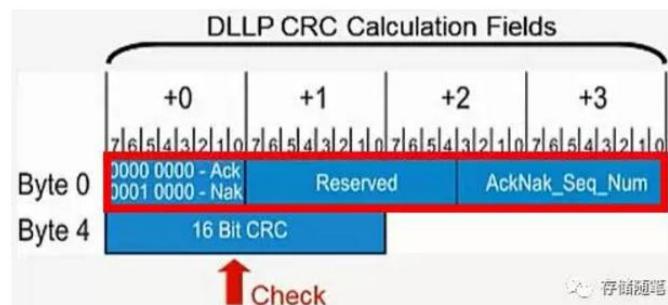


9. 发送端检查接收端返回的 DLLPs

当发送端收到接收端返回 Ack/Nak DLLPs 之后，会先检查其 CRC，如下图红色圈内所示。

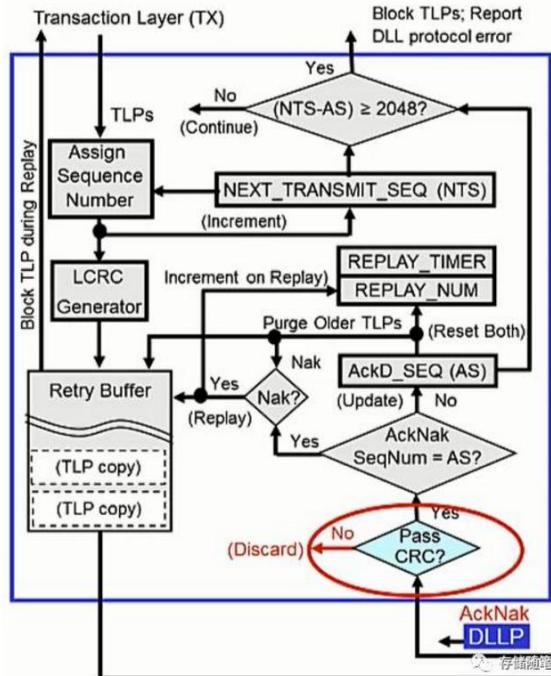


发送端会根据 Ack/Nak DLLPs 的 Byte0~3 计算 CRC，并与传进来的 CRC 作对比，验证是否一致。



(1) CRC 检查 fail

只要 CRC 检查 fail，当下的 Ack/Nak DLLPs 就会被舍弃。

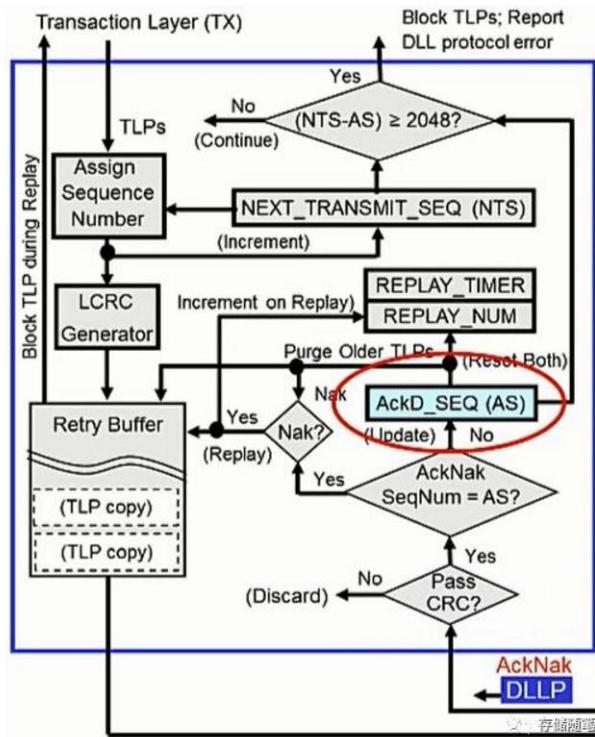


(2) CRC 檢查 OK 之後，繼續後續步驟

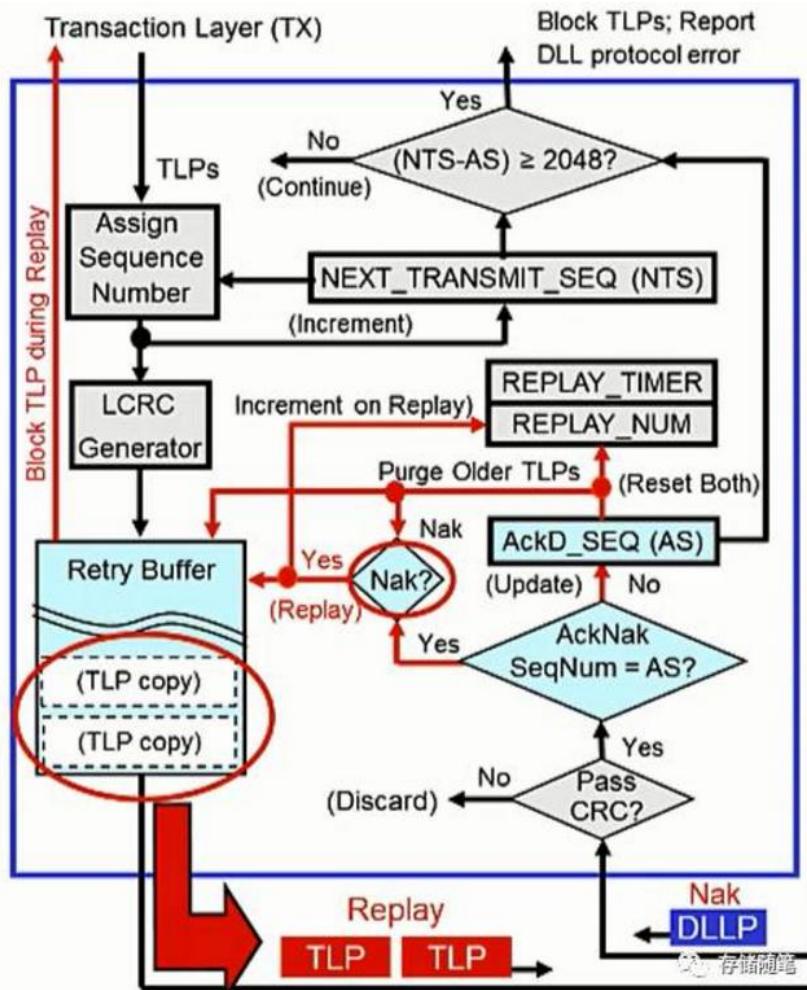
10. 發送端檢查 AS 參數

這裡有一個新的參數：Acknowledged Sequence Numbers，簡化標記為 AckD_SEQ，縮寫為 AS。AckD_SEQ 是一個 12 位的計數器，用於記載最近收到的 Ack/Nak DLLP 中的 Sequence ID。

當發送端收到的 Ack/Nak DLLP 中的 Sequence ID 大於 AS 時，代表 TLPs 传输正在进行中。



11. 接收到 Nak DLLP, TLP retry





当发送端接收到一个 Nak DLLP 时，代表前面传送的 TLP 有问题，需要重新发送。此时会有两种情况：

(1) 如果 Nak DLLP 的 Sequence ID=AS:

这个情况下，说明没有新的 TLP 传输。此时需要将 Retry buffer 中所有的 TLPs 重新发送，并且更新 Relay_NUM 加 1.

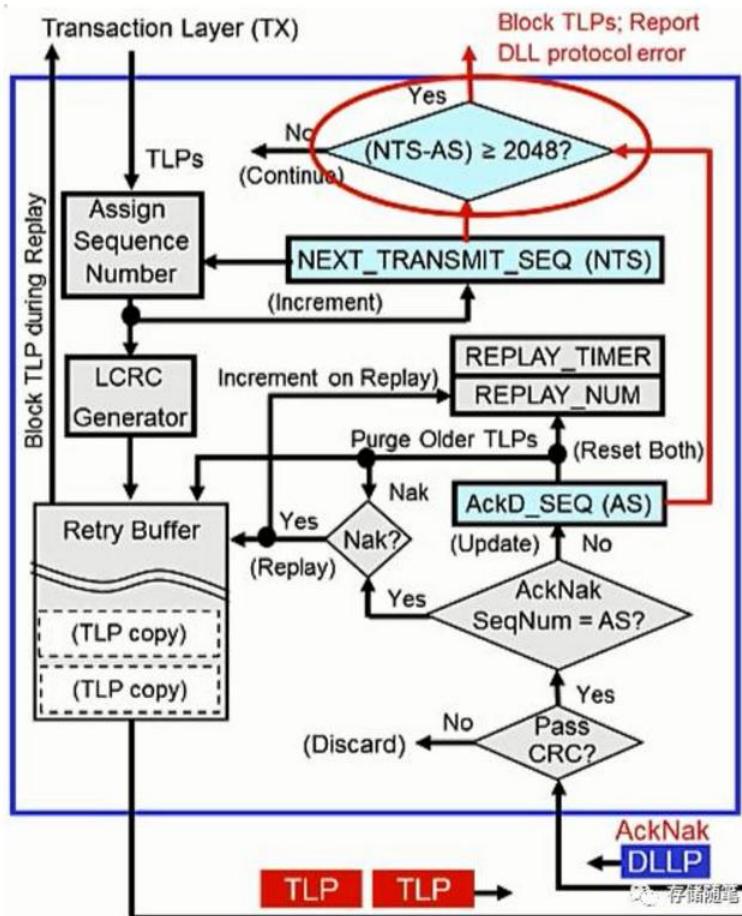
(2) 如果 Nak DLLP 的 Sequence ID>AS:

这个情况下，说明有新的 TLP 传输。此时需要将 Retry buffer 中 Sequence ID 之前的 TLP 全部清空，并将当下的 TLP 重新发送。与此同时，将 Replay_TIMER 以及 Replay_NUM 重置，並且 Replay_NUM 重置后加 1.

12. NTS-AS>=2048?

发送端在接收到 Ack/Nak DLLPs 最后一步要检验 NTS-AS 差值，NTS-AS 差值最小为 2048.

- ✧ 如果 NTS-AS>=2048 不成立，则说明数据链路层有协议错误。
- ✧ 如果 NTS-AS>=2048 成立，则数据链路层继续传输 TLPs。



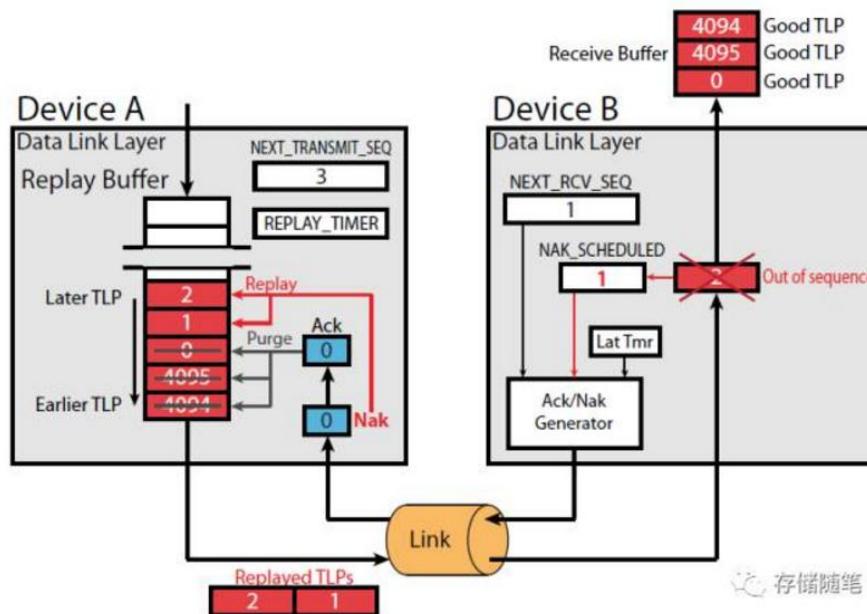
看完上面的理论之后是不是还有点晕晕的~我们再来看两个例子加深一下理解。

例 1: TLPs 丢失

- (1) 下图中，Device A 要给 Device B 传输 5 个 TLPs，Sequence ID 分别是 4094, 4095, 0, 1, 2.



- (2) TLP 4094 第一个被成功接收, 返回 Ack DLLP 给 Device A, 同时 Next_RCV_SEQ 加 1(也就是=4095)。
- (3) TLP 4095 第二个被成功接收, 返回 Ack DLLP 给 Device A, 同时 Next_RCV_SEQ 加 1(也就是=0, 因为 4095+1 超过了 Next_RCV_SEQ 的最大取值 4095, 从 0 开始记)。
- (4) TLP 4094 第三个被成功接收, 返回 Ack DLLP 给 Device A, 同时 Next_RCV_SEQ 加 1(也就是=1)。
- (5) 在 TLP 0 被成功接收之后, AckNak_LATENCY_TIMER 超时, 重新发送 Sequence ID=0 对应的 Ack DLLP。
- (6) TLP1 在传输的过程中由于某些原因(比如物理层的 Error)丢失, TLP 2 继续传输。但是在 Device B 端在等待 TLP 1。比较 Sequen ID (=2) 与 NRS(=1) 发现, Sequence ID>NRS, Device B 端才知道有 TLP 丢失。此时将 TLP 2 丢弃, 并回报 NRS-1(1-1=0) 对应的 Nak, 也就是 Nak 0.
- (7) Device A 端接收到 Sequence ID=0 对应的 Ack DLLP 之后, 重新发送 TLP 1 和 TLP2, 并且将 Sequence ID=0 之前的 TLPs(4094, 4095, 0) 全部从 Retry Buffer 里面清除。

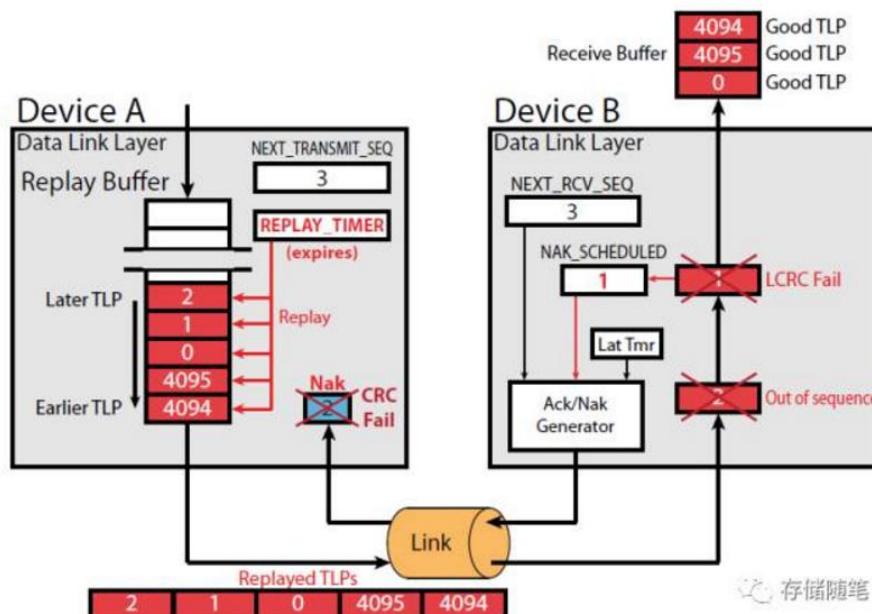


例 2：Nak DLLP 错误

- (1) 下图中, Device A 要给 Device B 传输 5 个 TLPs, Sequence ID 分别是 4094, 4095, 0, 1, 2.
- (2) TLP 4094, 4095, 0 被依次成功接收, 此时 Next_RCV_SEQ=1。
- (3) 在 TLP 1 到达 Device B 端, 检查 32-bit LCRC fail. 此时, 返回 NRS-1 对应的 Nak DLLP, 也就是 Nak 0.
- (4) Nak 0 到达 Device A 端, 检查 16-bit CRC fail, Nak 0 则被丢弃。
- (5) Nak 0 被丢弃后, Device B 不会再发送任何 Ack 或者 Nak。由于长时间没有收到 Device B 的反馈(Ack/Nak), Replay_TIMER 超时, Device A 将 Retry buffer 中所有的 TLPs 重新发送。



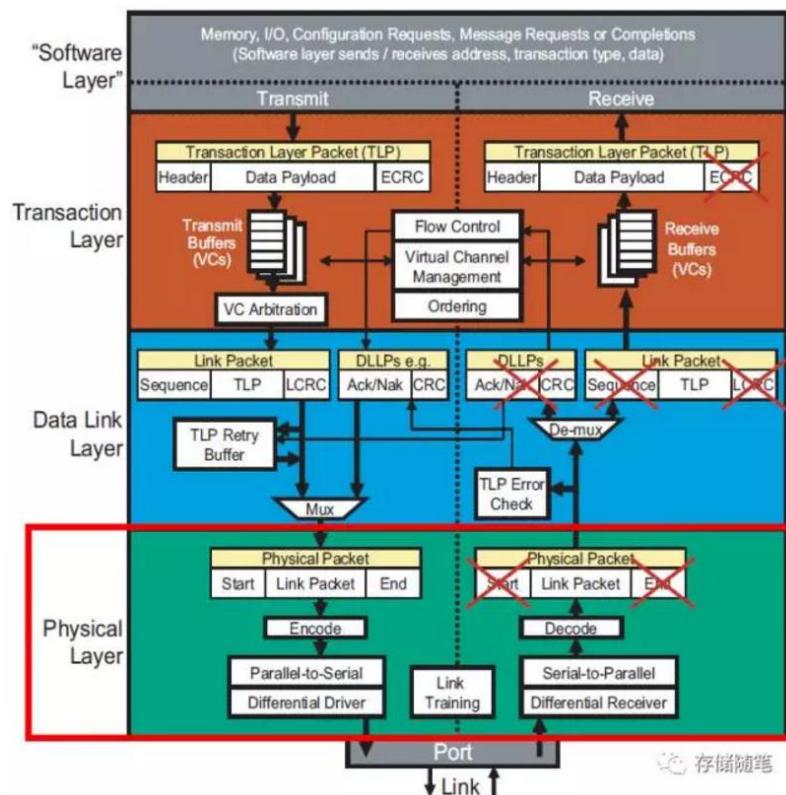
(6) TLP 4094, 4095, 0 重新发送后在 Device B 端会被识别到时重复的 TLPs，然后被丢弃。TLP1, 2 继续正确传输。



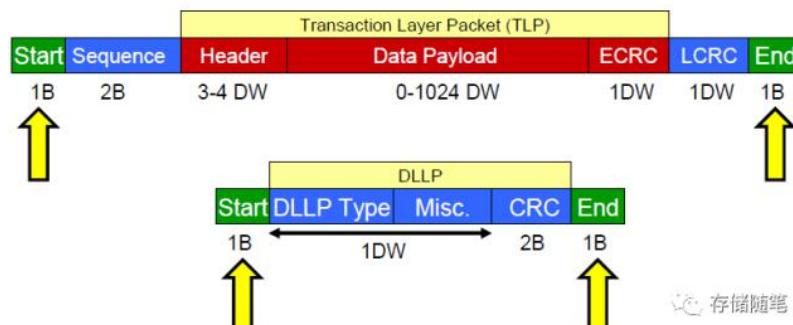
存储随笔

4.0 物理层结构解析

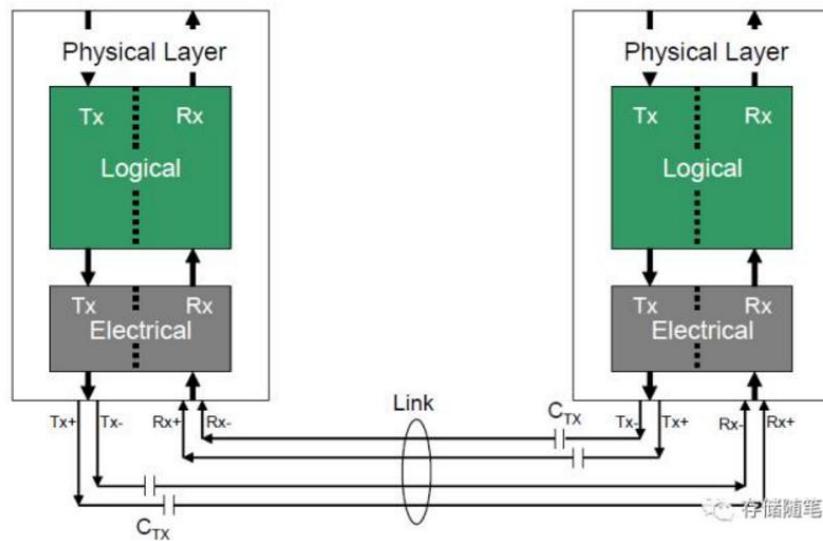
在 PCIe 体系中，物理层处于最底层。发送端数据链路层 (Data Link Layer) 的 DLLP 和 TLP 报文通过**物理层 (Physical Layer)**发送至接收端的**物理层**，再传送至接收端的数据链路层。



DLLP 和 TLP 从数据链路层到达物理层后，物理层会在其两端分别加上 Start 和 End 标识，主要是方便接收端找到 DLLP 和 TLP 的边界。



另外，物理层又分为两层：逻辑层(Logical)和电气层(Electrical)。逻辑层主要负责与数据链路层之间的数据交互，由发送逻辑 Tx 和接收逻辑 Rx 组成。电气层是物理层的模拟接口，包括了差分信号驱动和接收器。本专题对电气层不作展开介绍，主要会针对逻辑层进行解析。

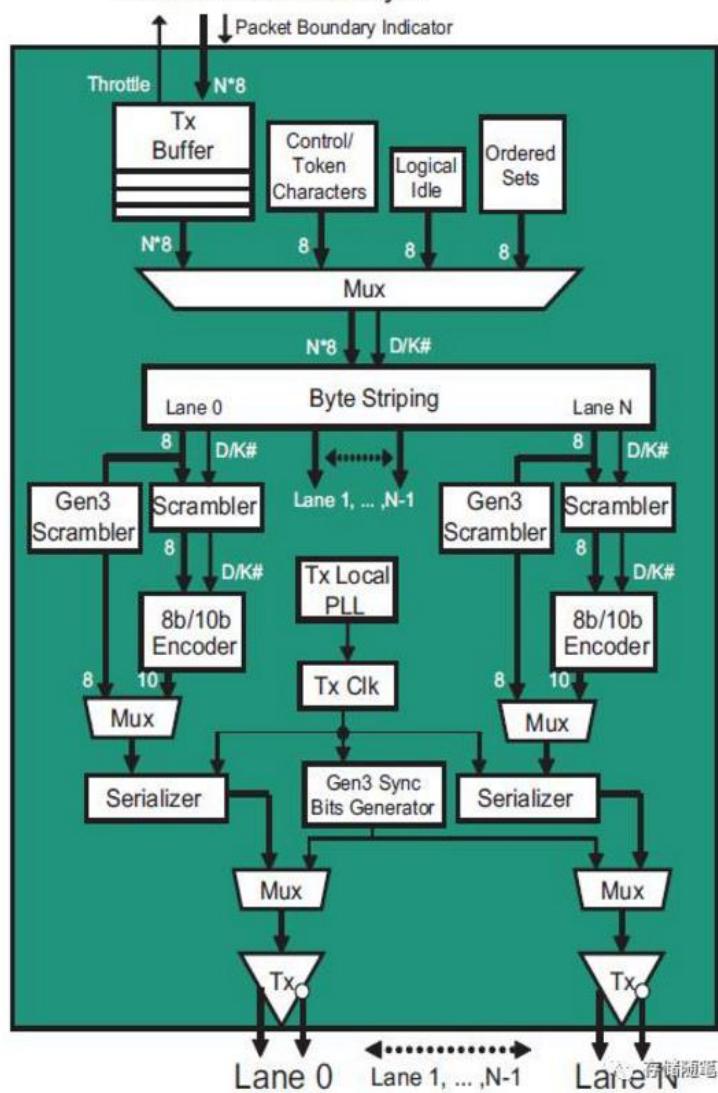


话说现在的社会是拼颜值的时代，那我们是不是先看一下逻辑层的样子呀？

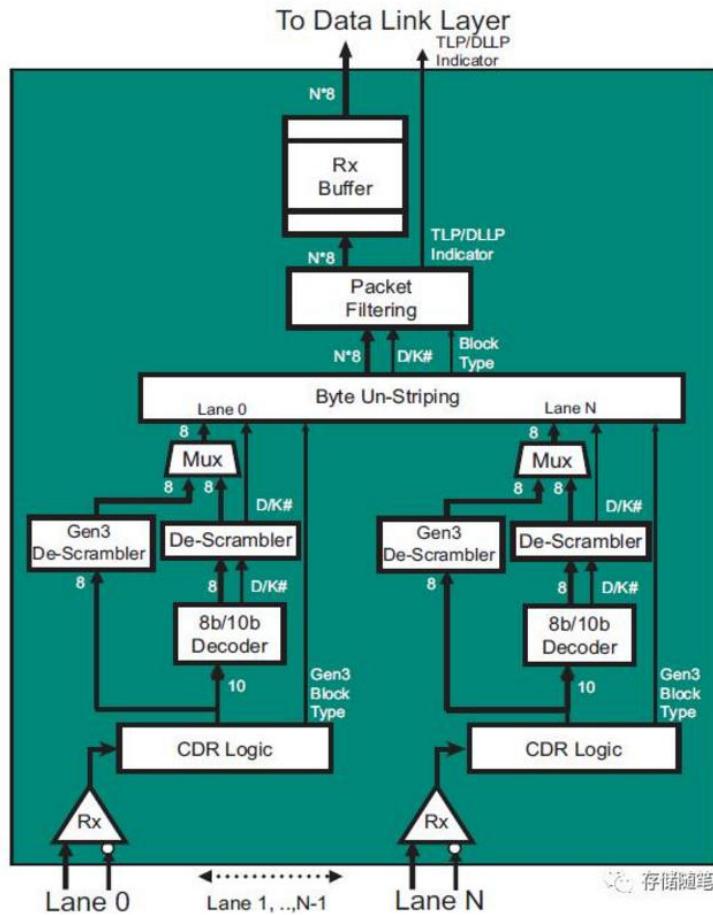
发送端逻辑层：



From Data Link Layer



接收端逻辑层：



看到上面两张逻辑层结构图，又晕了~

这里先结合上面的逻辑层结构图大致表述一下逻辑层的作用：

发送端：

- 从发送端数据链路层下发的 DLLP/TLP 在到达物理层后，会先放入 Tx Buffer 中。在 Tx buffer 中，DLLP/TLP 被加上前缀 Start 和后缀 End。
- 之后，DLLP/TLP 通过多路选择器 Mux，到达 Byte Stripping 组件。由于 PCIe 总线可能包含多个 Lane，Byte Stripping 组件将 DLLP/TLP 数据报文按照数据依次分发到不同的 Lane。
- 数据进入每个 Lane 之后都会做加扰(Scramble)，8/10b 编码(Only for Gen1/2)，128b/130b 编码(Only for Gen3)。
- 完成加扰和编码后，数据通过并转串逻辑，最后发送到 PCIe 链路中。

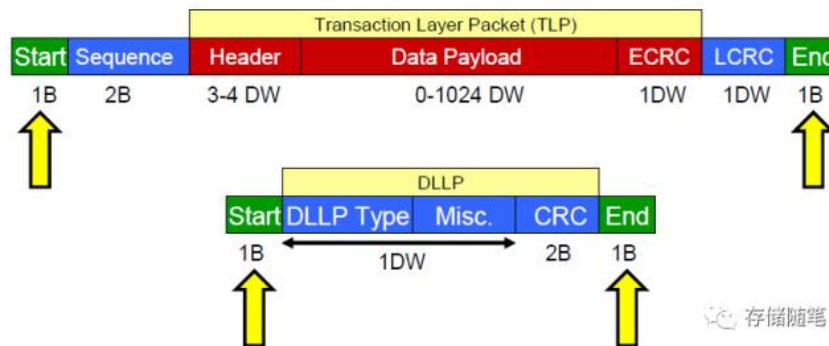
接收端：

- 接收端的逻辑层从 PCIe 链路的各个 Lane 中获得串行数据。
- 各个 Lane 的数据经过解码与解扰(De-Scramble)，最终到达 Byte Un-Stripping 组件。
- Byte Un-Stripping 组件将来自各个 Lane 的数据进行合并，检查 Start 和 End 标识后送入 Rx Buffer，最后传送至接收端数据链路层。



4.1 物理层数据流解析

上一篇文章中，我们有提到“DLLP 和 TLP 从数据链路层到达物理层后，物理层会在其两端分别加上 **Start 和 End 标识**，主要是方便接收端找到 DLLP 和 TLP 的边界”。



上面说到的 Start 和 End 标识，是物理层定义的控制字符中的两种。由于 Gen1&Gen2 (8b/10b 编码)与 Gen3 (128b/130b 编码)在物理层中的数据编码原理不同，物理层对 Gen1&Gen2 与 Gen3 定义的控制字符也不同。同样，不同数据编码也造就了不同的数据流格式。所以，接下来我们对 Gen1&Gen2 和 Gen3 的数据流分开解析。

由于与 SATA 专题介绍的 8b/10b 编码原理类似，PCIe 专题中将不再阐述 8b/10b 编码，感兴趣的话请参考：

[SATA 系列专题之二：2.1 Link layer 链路层 8b/10b 编码解析](#)

1. Gen1&Gen2 数据流

物理层针对 Gen1&Gen2 定义的控制字符主要有以下几类：

控制字符	说明
STP	=Start TLP， 代表 TLP 起始标志
SDP	=Start DLLP， 代表 DLLP 起始标志
END	TLP/DLLP 结束标志
EDB	=End Bad， 无效 TLP 结束标志
SKP	=Skip， 用于补偿 PCIe 链路中的延时
COM	=Comma， 将 PCIe 链路 Scramble 用的 LFSR 值初始化
IDL	=Idle， PCIe 链路进入 Electrical Idle 标识
PAD	数据流中的填充字符，无实际含义

除了上述的控制字符之外，还需要特别介绍一下 Ordered Sets。Ordered Sets 不是 TLP/DLLP，可以把 Ordered Sets 当做是 Lane 管家，比如 Link Training,



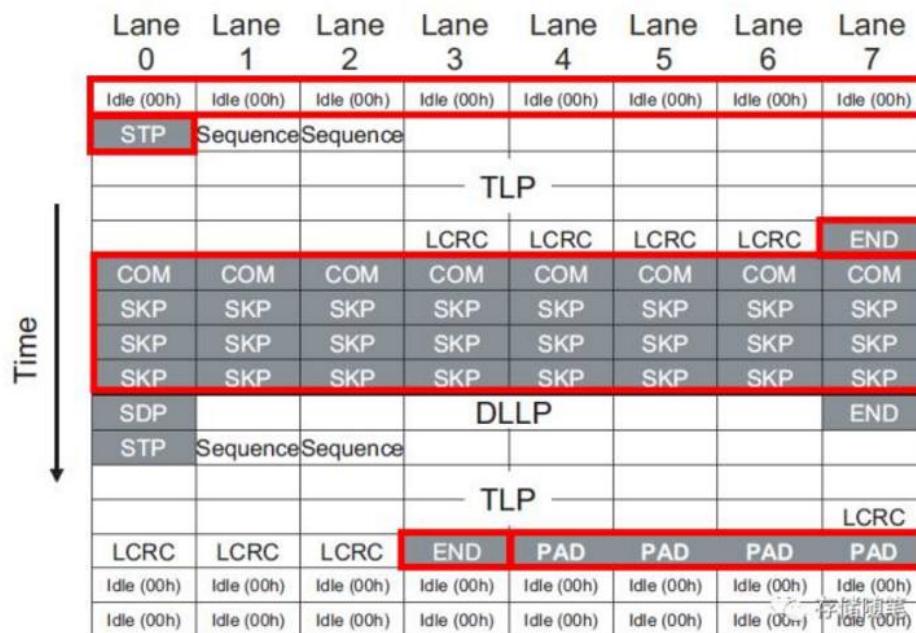
Link 电源管理等。在 Gen1&Gen2 中，以 COM 控制字符开头，所有的 Lane 必须同时发送 Ordered Sets。具体类型如下表：

Ordered Set (OS)	说明
TS10S/TS20S	用在 Link Training
EIOS	Electrical Idle，在传输结束之前发送该序列，让 PCIe 链路进入空闲状态。
FTSOS	告知 PCIe 链路从低功耗状态(L0s)进入正常工作状态(L0)
SOS	SKP OS，用于时钟补偿
EIEOS	Exit Electrical Idle，PCIe 链路退出空闲状态

无规矩不成方圆！说了控制字符和 Ordered Set，该说一下物理层中 Gen1 和 Gen2 数据流的一些规则了。

- ✧ 如果 PCIe 链路从 Logical Idle 之后开始数据流的传输时，STP 和 SDP 必须放在 Lane0；
- ✧ 如果不是从 Logical Idle 之后开始数据流传输，STP 和 SDP 可以放在 Lane0, 4, 8 等；
- ✧ 在 PCIe x2 链路中，END/EDB 放在 Lane1，其他链路中放在 Lane3, 7, 11 等；
- ✧ DLLP 数据包长度为 8 个字符，SDP+6 字符+END；
- ✧ 一个数据包结束之后，其他的数据包还没 Ready，这个时候需要 PAD 字符补位到最后一个 Lane；
- ✧ 当数据包传输结束，所有 Lane 发送 Logical Idle 字符“idle(00)”；
- ✧ 所有的 Lane 同时发送 Ordered Sets.

我们来看一下 PCIe Gen2 x8 的数据流，数据流里面包含了两个 TLP，一个 DLLP：





对照前面提到的数据流规则，上面这个例子就完全符合；

大家来找茬 1：

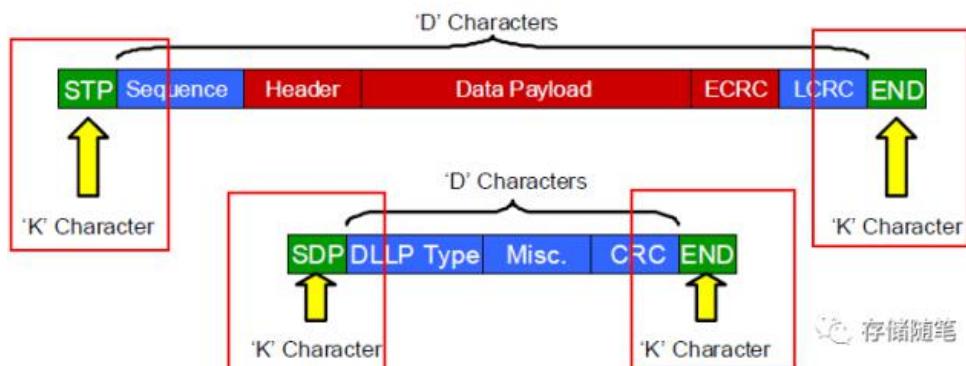
看看下面 PCIe Gen2 x8 的数据流有何错误，可以在文章底部简要作答哈~

Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
Idle	Idle	Idle	Idle	Idle	Idle	Idle	Idle
Idle	Idle	Idle	Idle	STP	Sequence	Sequence	
TLP							
LCRC	LCRC	LCRC	PAD	PAD	PAD	PAD	LCRC
COM	COM	COM	COM	SKP	SKP	SKP	SKP
SKP	SKP	SKP	SKP	SKP	SKP	SKP	SKP
SKP	SKP	SKP	SKP	SKP	SKP	SKP	END
SDP	DLLP						SDP

2. Gen3 数据流

在 Gen3 中，数据编码舍弃了原来 8b/10b 编码，而采用更加有效的 128b/130b 编码。这也改变了 Gen3 数据流的模式。

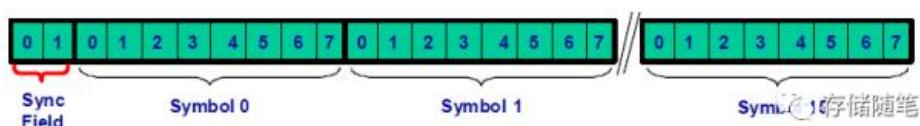
当 Gen1&Gen2 采用了 8b/10b 编码时，数据包前后会加上控制符 STP/SDP 和 END，格式如下：



而在 Gen3 采用 128b/130b 编码时，引入了一个新的概念：“块”(Block)；

一个 Block (130bits) = 2-bit Sync 字段 + 16Bytes

(128bits)



其中，Sync 字段有两个定义：

a, Sync 字段=01，代表是数据块(Data Block)；

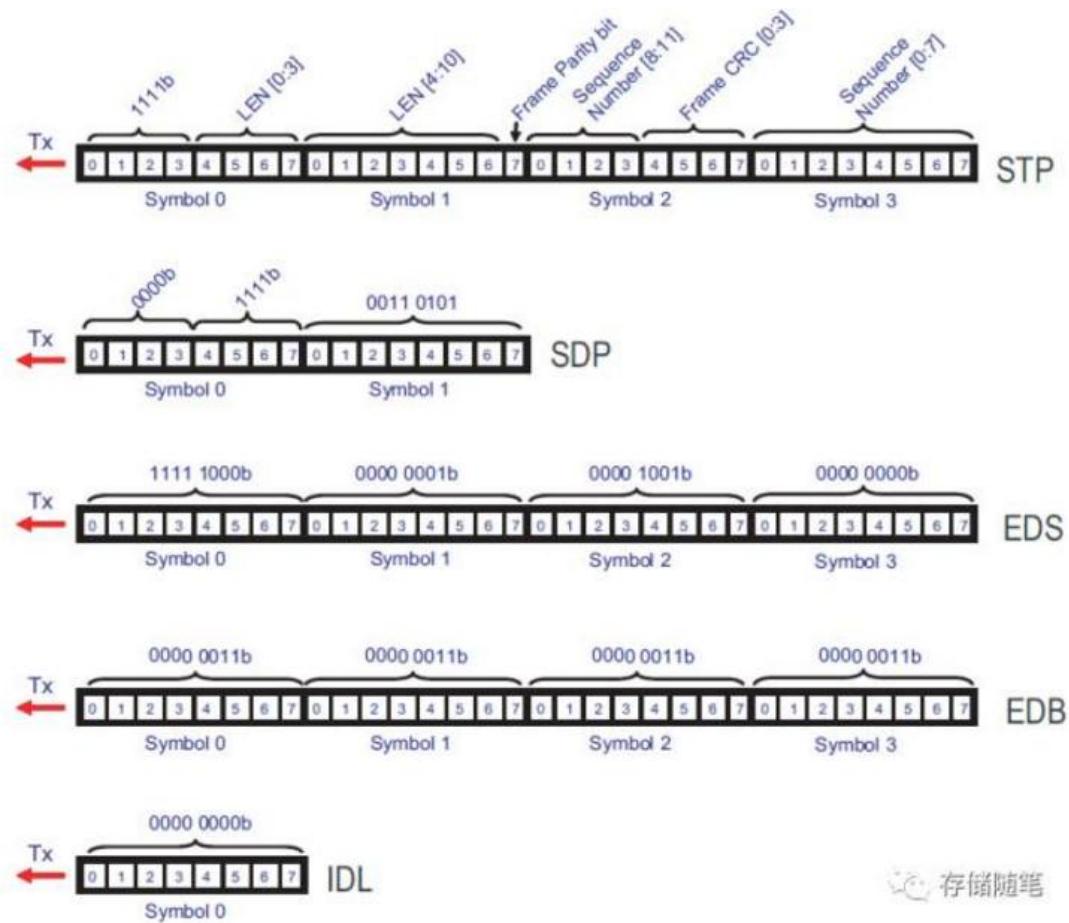
b, Sync 字段=10，代表是序列块(Ordered Set Block)。



物理层对 Gen3 数据块定义了 5 个字符(Token)，类似于 Gen1&Gen2 中的控制字符：

Token	大小	说明
STP	4 Bytes	=Start TLP, 代表 TLP 开始标识
SDP	2 Bytes	=Start DLLP, 代表 DLLP 开始标识
IDL	1 Bytes	=Logical Idle, 空闲状态
EDS	4 Bytes	数据流结束标识
EDB	4 Bytes	无效数据流标识

上述 Tokens 结构图如下：



此外，Gen3 Orderer Set 的定义与 Gen1/2 基本一致。

我们接下来看几个 PCIe Gen3 x8 数据流的例子：

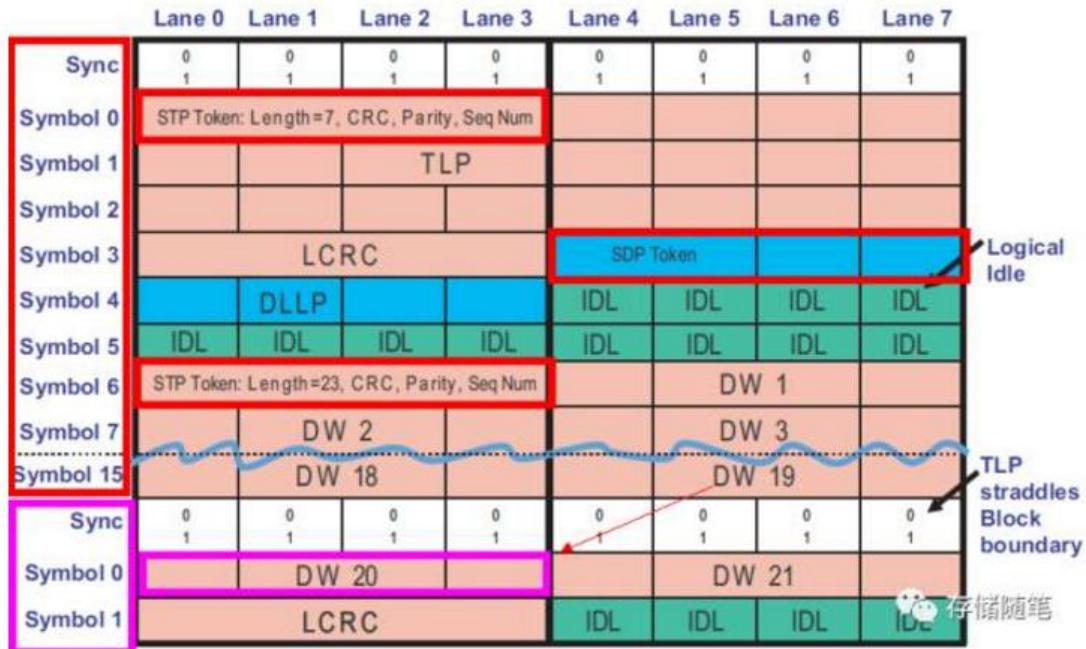
(1) 数据可以跨数据块(Block)传输

下面这幅图中的信息需要注意几点：

- ✧ STP 在 Idle 之后必须从 Lane0 开始；

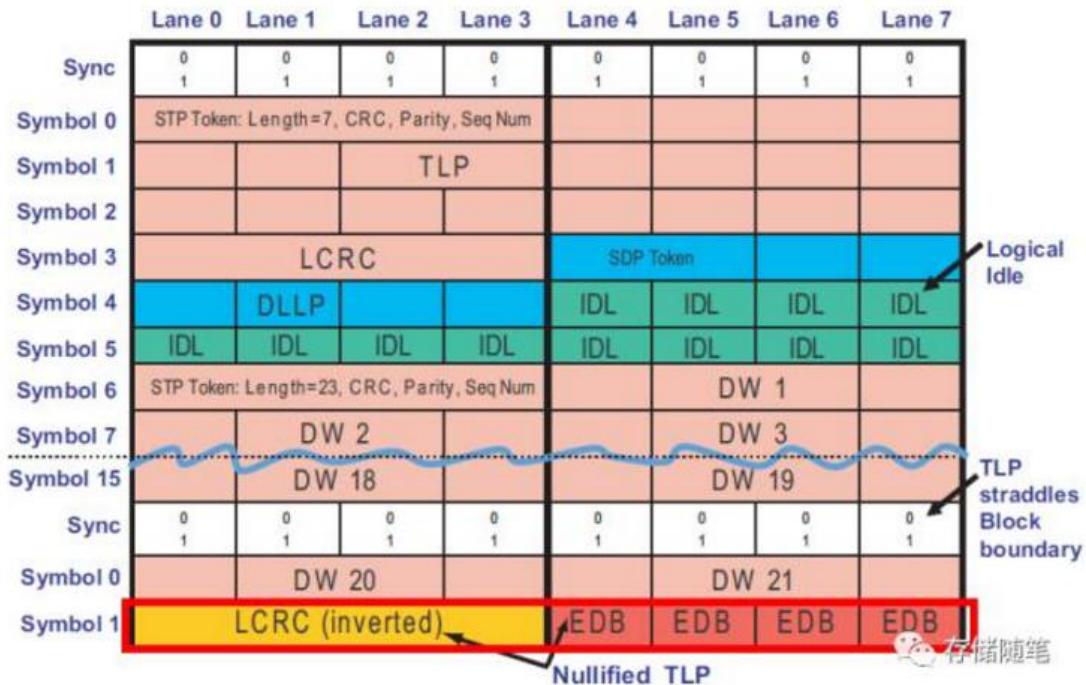


- 一个数据块=2-bit Sync+ Byte0~15;
- 一个数据块中未传完的数据可以在下一个数据块中接着传输，不用再发送 STP Token；



(2) 无效数据块传输

在上一个数据流的基础上，我们假设第二个 TLP 传输的过程中 LCRC 错误。那么 TLP 末端就会被加上 EDB 字段，表明此 TLP 已无效。

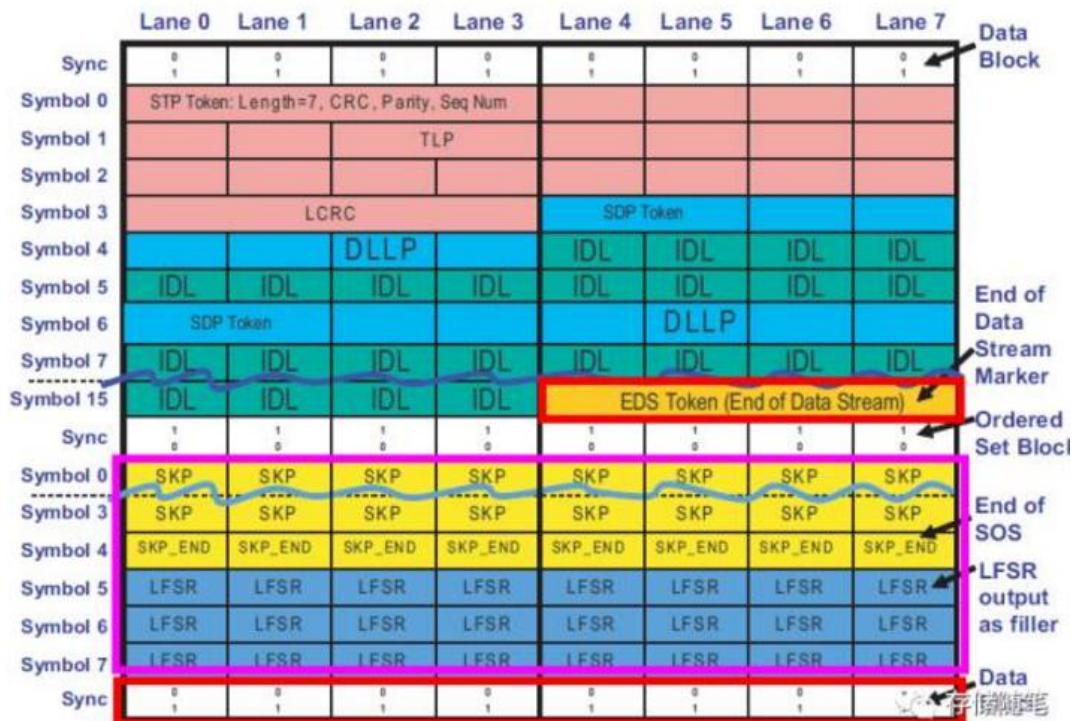


(3) 序列块(Order Set Block) SOS 传输

针对序列块 SOS(SKP OS)有以下几个规则：



- ✧ SOS 前面的数据块必须以 EDS 作为结束标志；
- ✧ 每个 Lane 要同时发送 SOS，SOS 大小一般为 16Bytes (12 SKP + 1 SKP_END + 3 Scramble 数据)
- ✧ 如果在 SOS 之后需要继续数据传输，必须以数据块开始。



大家来找茬 2：

看看下面 PCIe Gen3 x8 的数据流有何错误，可以在文章底部简要作答哈~

	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7	
Sync	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 0	
Symbol 0	STP Token: Length=23,CRC,Parity,Seq								
Symbol 1									
...									
Symbol 11									
Symbol 12	LCRC				SDP Token		DLLP		
Symbol 13	DLLP				IDL	IDL	IDL	IDL	
Symbol 14	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL	
Symbol 15	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL	
Sync	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 0	
Symbol 0	SKP	SKP	SKP	SKP	SKP	SKP	SKP	SKP	
...									
Symbol 11	SKP	SKP	SKP	SKP	SKP	SKP	SKP	SKP	
Symbol 12	SKP-END	SKP-END	SKP-END	SKP-END	SKP-END	SKP-END	SKP-END	SKP-END	
Symbol 13	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	
Symbol 14	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	
Symbol 15	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	LFSR	
Sync	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	



5.0 PCIe 总线电源管理

PCIe 总线的电源管理包括两方面的内容：

- ✧ 一是基于软件控制的 PCI-PM 电源管理机制，这部分与 PCI 总线兼容；
- ✧ 二是基于硬件控制的 ASPM 电源管理机制，不需要 Host 端软件口控制，PCIe 链路自主管理。

注：ASPM= Active State Power Management.

1. PCI-PM 电源管理

PCI-PM 电源管理机制是系统软件通过修改寄存器中的电源管理字段，使 PCIe 设备进入 D 状态。D 状态总共有四种：D0，D1，D2，D3。

- ✧ **D0**: 这个状态属于“全马力”工作状态，不考虑任何电源节省的因素。D0 又有两个子状态：D0-Uninitialized 和 D0 Active。
 - a, D0-Uninitialized: 这时 PCIe 还没有被激活，只能接受 Configure Write/Read TLP 请求，仍不能正常工作。比如设备刚被 Reset 后进入 D0-Uninitialized。
 - b, D0 Active: PCIe 设备已经被成功激活，可以正常工作。
- ✧ **D1**: PCIe 设备进入“浅睡眠”状态；
- ✧ **D2**: PCIe 设备进入“深度睡眠”状态；
- ✧ **D3**: 这个状态是电源管理中最低功耗的状态。有两个子状态：D3-hot 和 D3-cold。
 - a, D3-hot: 此时与 D1/D2 的功能类似，但是 D3-hot 只能返回到 D0-Uninitialized。
 - b, D3-cold: 当 PCIe 设备的 VCC 电源被移除时，PCIe 设备进入此状态。

值得注意的是，**PCIe 设备必须支持 D0 和 D3**，但是 D1/D2 是选择性的。所以在目前市面的 SSD PCIe 主控，一般都只是支持 D0/D3，并不支持 D1/D2。当 PCIe 设备处于 D0 状态时，ASPM 可以改变 PCIe 链路的电源状态。

介绍 ASPM 之前，我们先认识一下 PCIe 定义的有关 PCIe 链路的电源状态：

- ✧ **L0**: 这个状态属于 PCIe 设备的工作状态，与 D0 对应；
- ✧ **L0s**: PCIe 设备进入 Standby 状态；
- ✧ **L1**: PCIe 设备进入比 L0s 更低功耗的 Standby 状态；L1 状态有两个子状态 L1.1 和 L1.2；
- ✧ **L2/L3 Ready**: 这两个状态是 PCIe 设备准备进入 L2/L3 前的预备状态；
- ✧ **L2**: 比 L1 功耗更低的深度省电状态；
- ✧ **L3**: 此时 PCIe 链路出于关闭状态，PCIe 设备的 VCC 电源也被移除；



- ◆ LDn: 这个状态不具有实际意义，只是 L2/L3 返回 L0 状态时所需要的中间过渡状态；

PCIe 设备的 D 状态与 PCIe 链路电源状态相辅相成，不是单独存在的。亲密关系如下表：

Downstream Component D-State	Permissible Upstream Component D-State	Permissible Interconnect State
D0	D0	L0, L0s & L1 (optional)
D1	D0-D1	L1
D2	D0-D2	L1
D3 hot	D0-D3 hot	L1, L2/L3 Ready
D3 cold	D0-D3 cold	L2 (AUX 1-wf), L3

了解了 PCIe 设备 D 状态与 PCIe 链路的电源状态，我们接下来看看“一号主人公”ASPM。

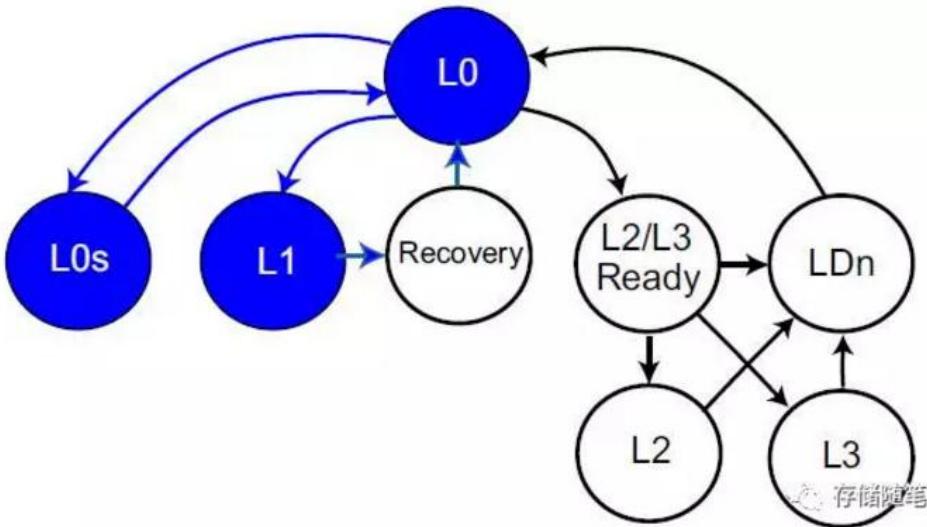
2. ASPM 电源管理

ASPM 是基于硬件自主控制的链路电源管理机制，只有在 PCIe 设备处于 D0 状态是才可以应用 ASPM 机制。与 ASPM 有关的链路状态只有 L0s 和 L1。

State	Description	Software Directed?	Active State Link PM	Ref. Clocks	Main Power	PLL	Vaux
L0	Fully Active	Yes (D0)	On	On	On	On	On/Off
L0s	Standby	No	Yes (D0)	On	On	On	On/Off
L1	Low Power Standby	Yes* (D1-D3 hot)	Yes (option) (D0)	On	On	On/Off	On/Off
L2/L3 Ready	Staging for power removal	Yes PME_Turn_Off handshake	No	On	On	On/Off	On/Off
L2	Low Power Sleep	Yes**	No	Off	Off	Off	On
L3	Off (Zero Power)	N/A	N/A	Off	Off	CPU 存储随笔	

在 L0s 和 L1 状态下，PCIe 链路均进入 Electrical Idle。此时，链路中差分信号 D+ 和 D- 没有压差，也就代表没有信号传输。Electrical Idle 状态的进入和退出会用到数据流的 Ordered sets (TS1/TS2, EIOS, FTS, EIEOS 等)，有关数据流的相关内容请见前面的文章“物理层数据流解析”，在这里不再展开介绍。

我们这里主要介绍一下 L0s 和 L1 的状态转换。



从上面 ASPM 链路状态转换图(蓝色部分)中，我们可以看到：

- ✧ L0s 与 L0 之间可以相互转换；
- ✧ L1 返回 L0 时不能直接返回，需要先进入 Recovery 状态；
- ✧ L0s 与 L1 之间不能直接转换，要切换状态，必须先回到 L0 状态。

L0s 状态的进入：

L0s 状态的进入是根据设备所处空闲时间的长短决定的。PCIe Sec 规定，如果 PCIe 设备在 7us 之内，没有 TLPs 或者 DLLPs 要传输，那么 PCIe 设备发送端就可以决定是否要进入 L0s 状态。

不过，这个不是强制性的，如果你任性的坚持即使超过 7us 没有 TLPs/DLLPs 传输也不进入 L0s，也是可以接受的~但这样的话就失去了 ASPM 的意义咯~

当 PCIe 设备发送端决定进入 L0s 时，发送端会停止数据输出，并给接收端发送 Ordered set "EIOS"，之后，发送端就进入 Electrical Idle 状态。接收端接收到 EIOS 序列之后，在经过一段延时(最短 20ns)后，也进入 Electrical Idle 状态。此时，发送端和接收端均进入 L0s 状态。

另外，需要提一点：发送端与接收端有可能出于不同的链路状态。比如，进行 DMA 操作时，发送端一直处于工作状态 L0，而接收端在长时间内无事可做，将自主进入 L0s 状态，此时不需要发送端同步进入 L0s。

L0s 状态的退出：

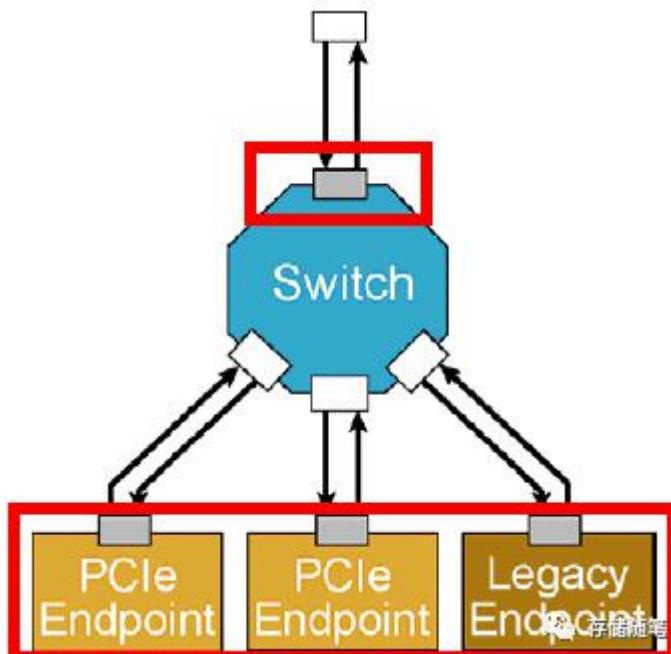
当 PCIe 设备的发送端检测到空闲状态的条件(7us 之内没有 TLPs 或者 DLLPs 要传输)不再满足时，就会从 L0s 退回 L0 状态。此时，发送端会向接收端发送 N_FTS(Link Training 时设定)个 FTS 和 1 个 SOS，之后进入 L0 状态。接收端收到满足要求的 FTS 后，再接收 1 个 SOS 序列，然后也进入 L0 状态。

与 L0s 相比，L1 的功耗更低。同时 L1 状态的进入与退出都比 L0s 要复杂。



L1 状态的进入：

L1 要求 PCIe 链路两端的设备均进入 L1 状态。重要的是，只有下游设备 (EP 活着 Switch 上游端口，如下图红色框) 可以主动要求进入 L1 状态。上游设备在接收到下游设备的请求之后，根据自身状态决定是否接收上游设备的请求。



其实，下游设备也不是很任性的随时可以请求进入 L1 状态，也需要满足几个条件：

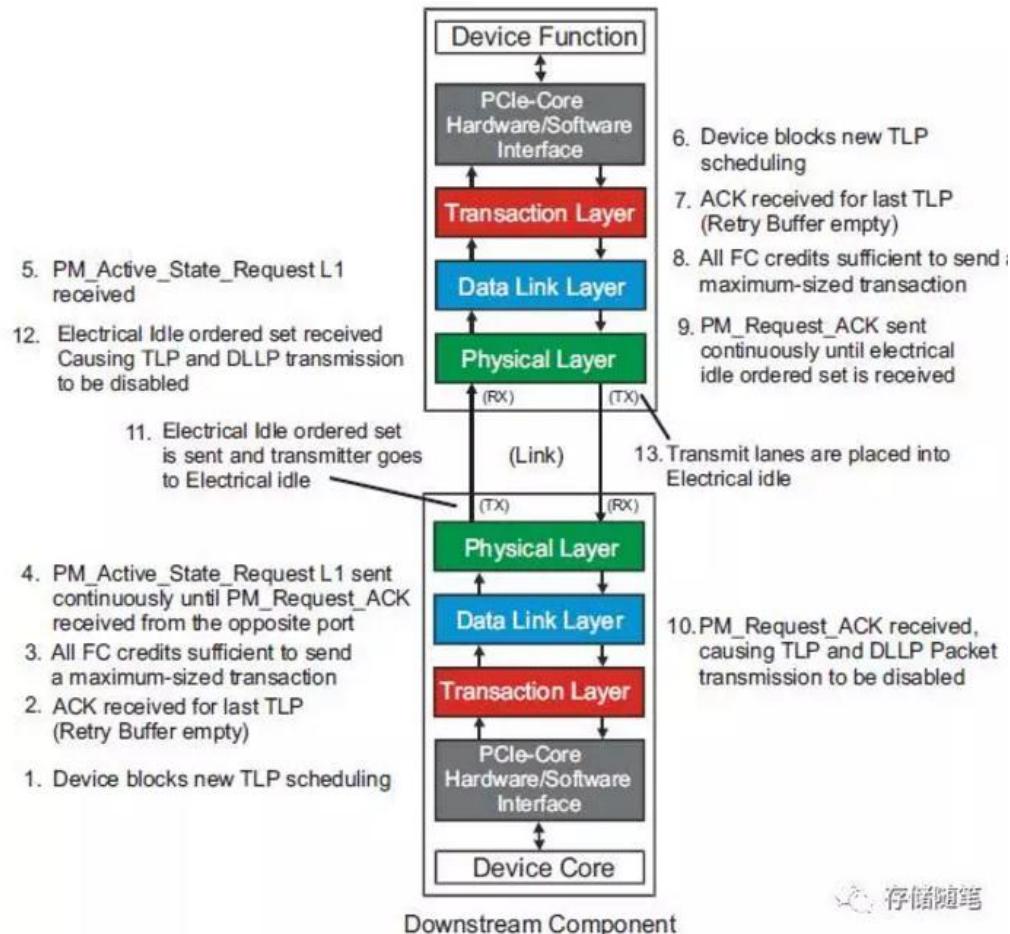
- ✧ PCIe 设备没有 TLP 和 DLLP 需要传输；
- ✧ PCIe 设备支持 L1 状态；
- ✧ 如果是 Switch 作为下游设备启动 L1 状态，前提是 Switch 下游端口必须处于 L1 或者比 L1 更高的省电状态(比如，L2/L3).

那么，我们就详细解读下游设备是如何启动请求进入 L1 状态的。

1. 当下游设备准备启动进入 L1 状态时，会先暂停安排新的 TLP；
2. 在发送进入 L1 状态请求之前，必须接收到暂停 TLP 传输前的最后一个 TLP 的 Ack 回报，确保其已被正确接收；
3. 另外，还要确保 Flow Control 积分单元可以满足最大 TLP 传输的需要。这个主要是为了设备退出 L1 状态之后可以立刻开始 TLP 传输工作；
4. 在前面的准备工作完成之后，此时就可以放心的给上游设备发送进入 L1 状态的请求咯。在这个过程中，下游设备会一直发送 PM_Active_State_Request L1 DLLP，知道收到对方的“接收函”PM_Request_ACK DLLP。
5. 上游设备接收到进入 L1 状态的请求。
- 6-8. 上游设备在接收到 L1 请求之后会重复类似上游设备的准备工作 (Step 1-3)。
9. 上游设备的准备工作完成后，会给下游设备下发“接收函”PM_Request_ACK DLLP，同意下游设备的请求。
10. 下游设备接收到上游的接收函之后，停止 TLP 和 DLLP 的传输。
11. 下游设备发送端发送 EIOS 序列，然后进入 Electrical Idle 状态。



12. 上游设备收到下游设备的 EIOS 序列后，知道下游设备已进入 L1 状态。此时，上游设备也停止 TLP 和 DLLP 的传输。
13. 接着上游设备的发送端也进入 Electrical Idle 状态，代表了上游设备也进入 L1 状态。



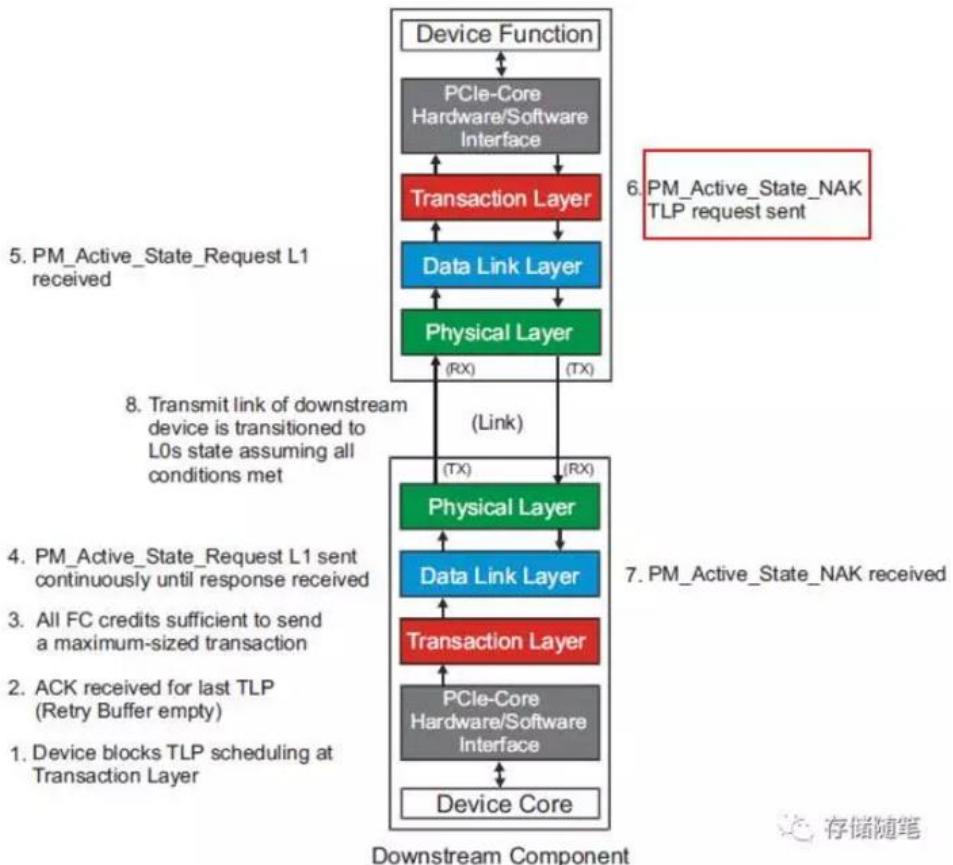
其实，上述的过程是一种比较理想的状态，就是假设在下游设备发送进入 L1 请求的时候，双方均已完成事务传输，并且没有新的事务传输需求。

生活不可能总是一帆风顺。下游设备的进入 L1 请求也有可能被上游设备拒绝：

1. 当下游设备准备启动进入 L1 状态时，会先暂停安排新的 TLP；
2. 在发送进入 L1 状态请求之前，必须接收到暂停 TLP 传输前的最后一个 TLP 的 Ack 回报，确保其已被正确接收；
3. 另外，还要确保 Flow Control 积分单元可以满足最大 TLP 传输的需要。这个主要是为了设备退出 L1 状态之后可以立刻开始 TLP 传输工作；
4. 在前面的准备工作完成之后，此时就可以放心的给上游设备发送进入 L1 状态的请求咯。在这个过程中，下游设备会一直发送 PM_Active_State_Request L1 DLLP，知道收到对方的“接收函”PM_Request_ACK DLLP。
5. 上游设备接收到进入 L1 状态的请求。
6. 上游设备在接收到 L1 请求之后，发现自身情况并不能满足 L1 的要求，并回复 PM_Active_State_NakTLP，告知下游设备目前无法进入 L1 状态。
7. 下游设备接收到上游的拒绝信。



8. 因为无法进入 L1 状态，下游设备会退而求其次，在满足 L0s 时先进入 L0s 状态。



存儲隨筆

L1 状态的退出：

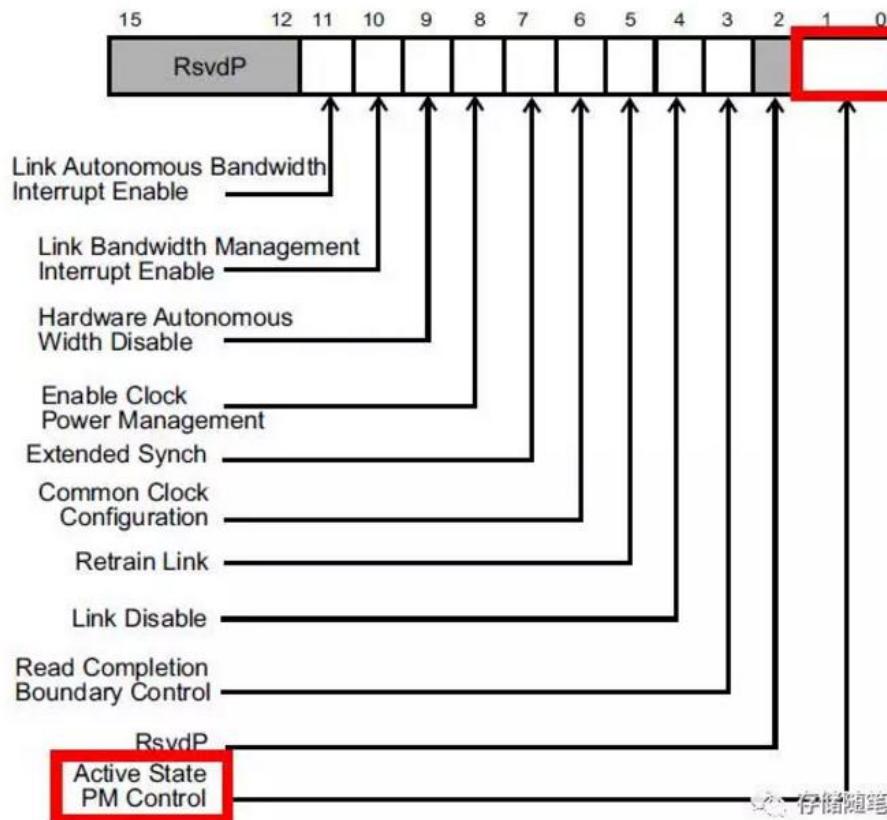
与 L1 状态的进入步骤相比，L1 状态的退出相对简单很多了。L1 状态的退出请求可以有任何一方提出。不像 L1 状态的进入，只有下游设备才能请求进入 L1 状态。

L1 状态的退出是通过重新 Link Training 实现的。因为发送 TS1 序列之后，设备退出 Electrical Idle 状态，也即退出 L1，进入 Recovery 状态，进而回到 L0 状态。

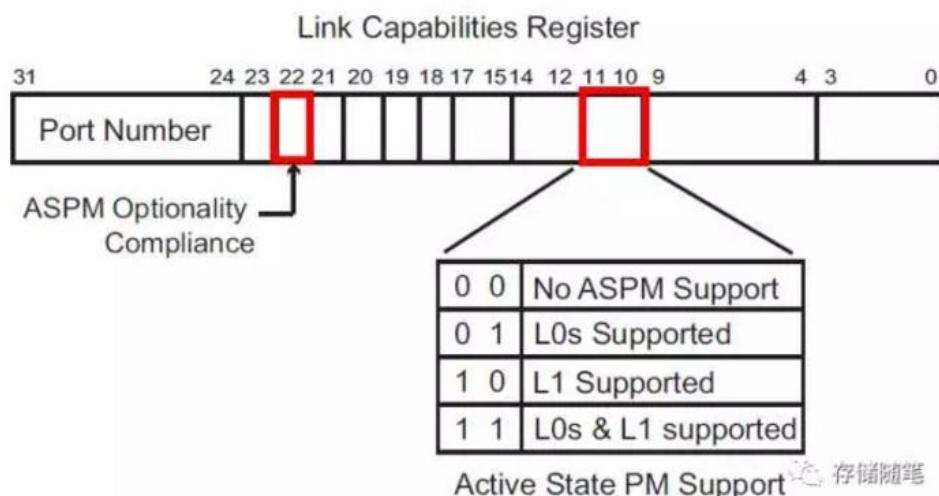
上面介绍了 ASPM 相关的链路状态，是不是觉得 ASPM 挺好玩的呢？

如果我们要在 SSD 中应用 ASPM 功能，其实还有两项工作：

(1) 在 Link Control Register 中，将 ASPM 的功能打开：



(2) 在 Link Capabilities Register 中，对 ASPM 进行具体的设定。



此外，除了上述通过软件修改 Link Control Register 的方式打开 ASPM 功能外，现在很多主板 BIOS 中可以修改 ASPM 的设定。



6.0 PCIe 系统复位方式

在 PCIe Spec 中，Reset 总共分为两类：Conventional Reset 和 Function Level Reset。

1. Conventional Reset

从字面上来讲，Conventional Reset 是传统的 Reset 方式。这一类 Reset 功能是在 PCIe Spec 2.0 之前的 Spec 中定义的，所以称为传统的 Reset。PCIe 设备必须要支持这一类 Reset。

Conventional Resets 包含了三种 Resets: Cold Reset, Warm Reset 和 Hot Reset。另外，还有一个概念：Cold Reset 和 Warm Reset 又被称为 Fundamental Reset，Hot Reset 被称为 Non-Fundamental Reset。

什么是 Fundamental Reset 呢？

这是 PCIe 最基本的复位方式，主要通过硬件实现，效果是重置整个设备，对每个状态机、所有硬件逻辑、端口状态和配置寄存器重新初始化。

但是，也会有例外的情况：在某些寄存器中的字段只有在全部电源（包括 VCC 电源和 Vaux 备用电源）切断的情况下才会被重置。PCIe Spec 给这些固执的字段起了个外号“Sticky Bits”。

一般来说，Fundamental Reset 是针对整个系统做 Reset，但是有时也可以针对某个单一设备进行重置。

在这里说明一下 Fundamental Reset 中的 Cold Reset 和 Warm Reset。

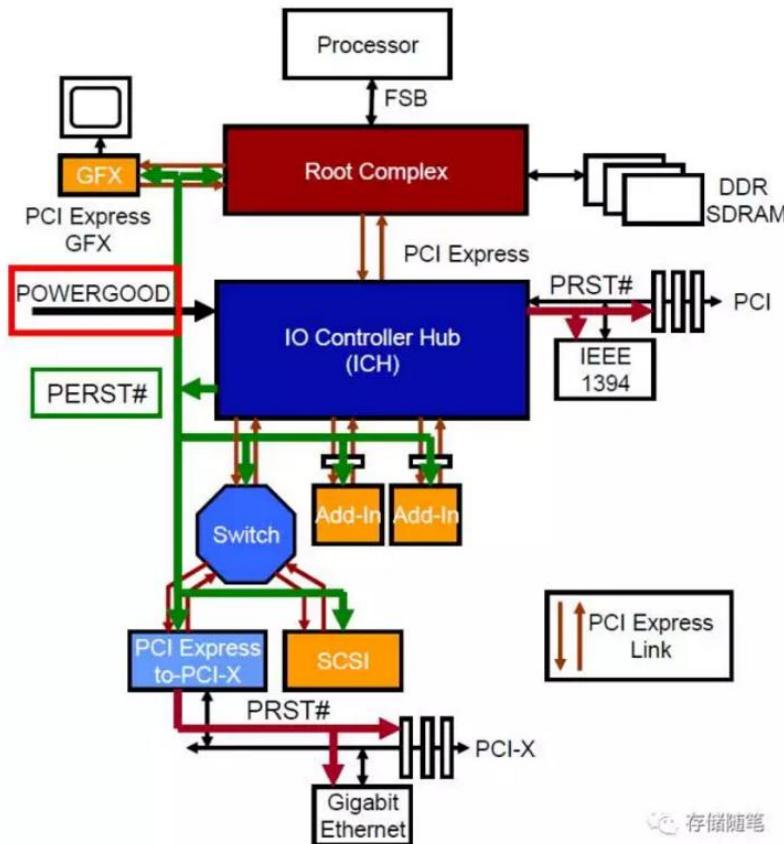
- ✧ **Cold Reset**: 设备的主电源 VCC 上电时，就会触发 Cold Reset。
- ✧ **Warm Reset**: 在 VCC 不断电的情况下，系统可以触发 Warm Reset。比如，电源状态的变化就会触发 Warm Reset。不过，PCIe Spec 并没有定义触发 Warm Reset 的具体方式，这部分可以由系统设计人员自行决定。

另外，在 PCIe Spec 中，规定了两种触发 Fundamental Reset 方式。

- ✧ 一是通过 PERST#(PCIe Reset) 信号控制。
- ✧ 二是在没哟 PERST#信号的情况下，通过 Power on/off 的方式实现。

举个例子，看看 PERST#是如何生成的。

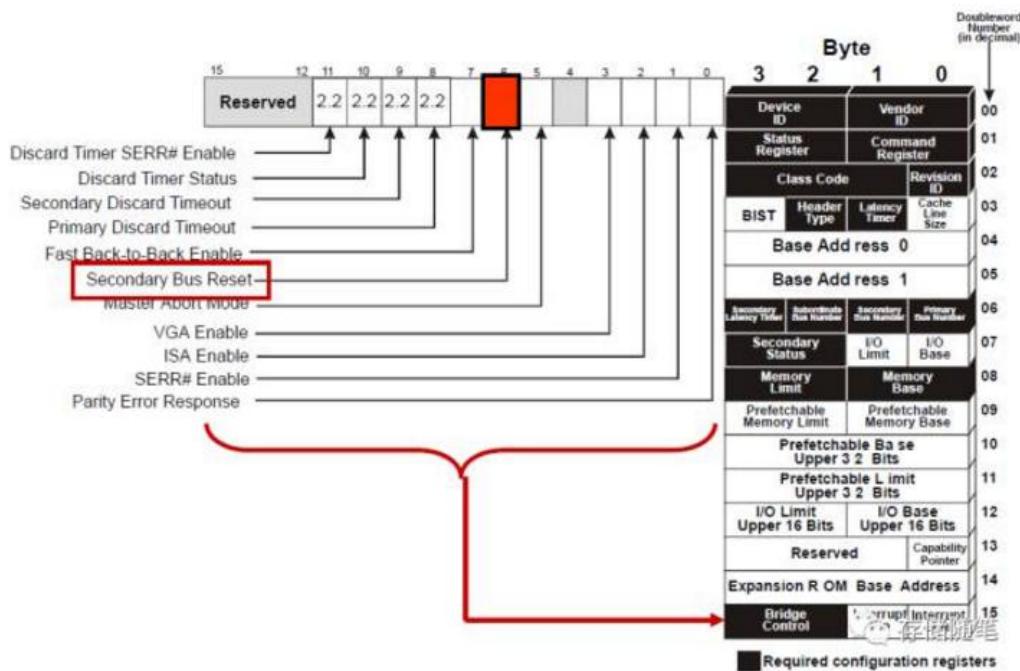
- 1) 系统上电稳定后，有 POWERGOOD 信号产生（下图红色框所示）。
- 2) 当系统的南桥芯片（也就是图中的 IO 控制器 ICH）收到 POWERGOOD 信号后，就会产生 PERST#信号（下图绿色部分），此时会引起 Cold Reset。
- 3) 如果系统可以通过非上电的方式触发 PERST#信号，此时会引起 Warm Reset。



存储随笔

明白了 Fundamental Reset，那 Non-Fundamental Reset 中的 Hot Reset 又是什么呢？

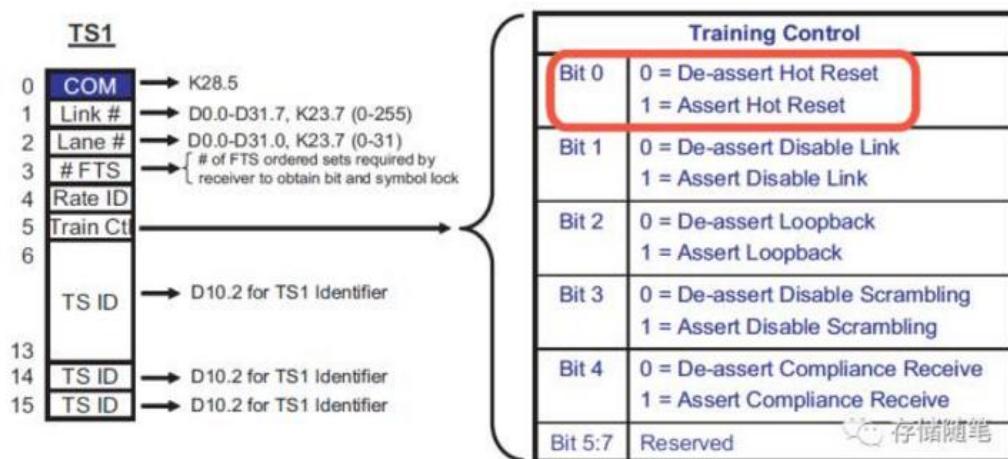
与 Fundamental Reset 相反，Hot Reset 是一种软件控制的复位方式。PCIe 设备出现错误时，通常情况下用软件的方式对设备重置。软件可以通过在 Bridge control 中设置 Secondary Bus Reset bit 来触发 Hot Reset。





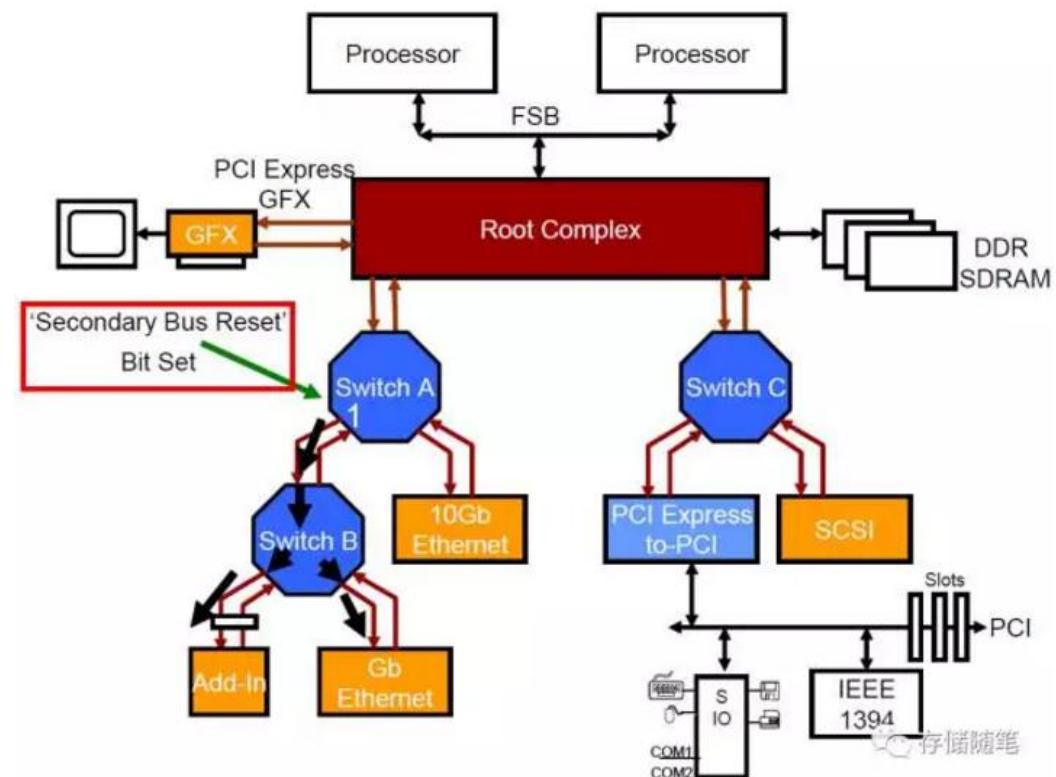
另外，在PCIe总线中，通过发送TS1序列，并且在TS1序列中设置Hot Reset bit来对下游设备进行Hot Reset(如下图红色框).

在这个过程中，发送端会持续发送TS1序列至2ms，接收端在接到2个连续的TS1序列之后进行Hot Reset.



同样，举个例子说明一下Hot Reset：

- 1) 系统通过软件对Switch A左边端口的Secondary Bus Reset bit置为1(下图红色框)，触发了Hot Reset。
- 2) 之后通过发送TS1序列对PCIe链路中的下游设备触发Hot Reset(下图黑色箭头)。





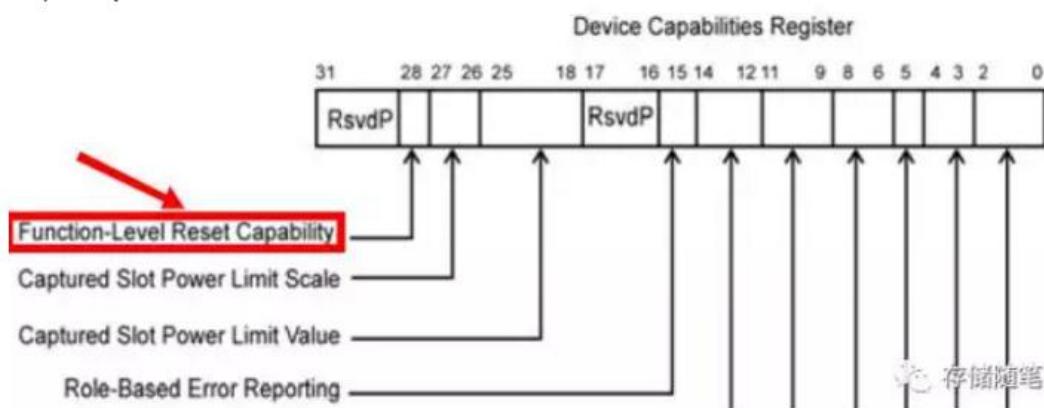
2. Function Level Reset

在传统复位方式的基础上，PCIe Spec 2.0 以后开始增加了新的复位方式 FLR (Function Level Reset)。前面讲到的传统复位方式 (Cold Reset, Warm Reset, Hot Reset) 均属于全局复位方式，而 FLR 的优势则是对局部复位。

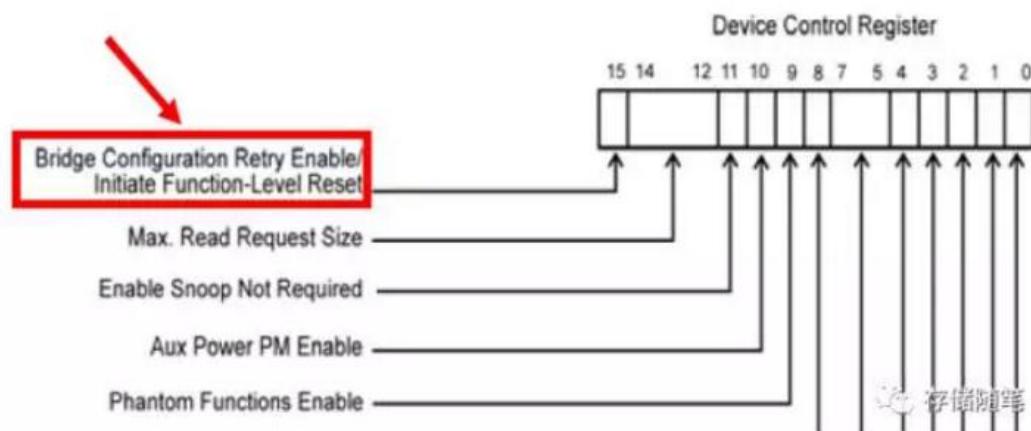
在 PCIe 协议中，一个 PCIe 设备可能包含多个功能模块 (Function)，每个功能模块相互对立，共用一个 PCIe link。其中，某个功能模块出问题时，虽然可以采用传统复位方式对整个 PCIe 设备复位，但这个显然不友好，因为其他功能模块可能正在埋头苦干。这就好比如，在一个团体中，一个人犯错了，要团队所有人一起承担，这个肯定会影响团结呀。

所以，PCIe 深得管理学的精髓，为了不影响团结，FLR 允许只对其中出错的功能模块 (Function) 进行重置，其他功能模块正常工作。

不过，FLR 复位方式对 PCIe 设备并不是必须的，在对 PCIe 设备使用 FLR 复位之前必须先检查是否支持 FLR。这部分可以查看 Device Capabilities Register 是否将 Function-Level Reset Capability bit 置起。



如果 PCIe 设备支持 FLR，那么就可以通过设置 Device Control Register 中的 Function-Level Reset bit 触发 FLR 复位咯~





触发 FLR 之后，PCIe 链路中都有哪些变化呢？

我们前面提到了，FLR 是一个局部复位方式，只对出问题的那个 Function 起作用。所以说，**FLR 只会改变当下 Function 内部的状态和寄存器的内容**。以下几个方面不会被影响：

- ✧ 执行 FLR 的 Function 所在的 PCIe 链路状态不会改变，因为其他 Function 也在共用整条 PCIe 链路；
- ✧ Sticky Bits. 传统复位方式也无法改变 Sticky bits，除非完全断电。
- ✧ HwInit Bits. HwInit bits 是硬件初始化的内容，这些值由芯片的配置引脚决定，后者上电复位后从 EEPROM 中获取。Cold 和 Warm Reset 可以复位这些寄存器，然后从 EEPROM 中重新获取数据，但是使用 FLR 方式不能复位这些寄存器。
- ✧ 与 Link 相关的寄存器。比如 ASPM, Flow control 等相应的寄存器。

另外，PCIe Sepc 规定，某个 Function 的 FLR 必须在 100ms 之后完成。所以，PCIe Spec 写了一封倡议书给要使用 FLR 复位方式的“人们”-软件：

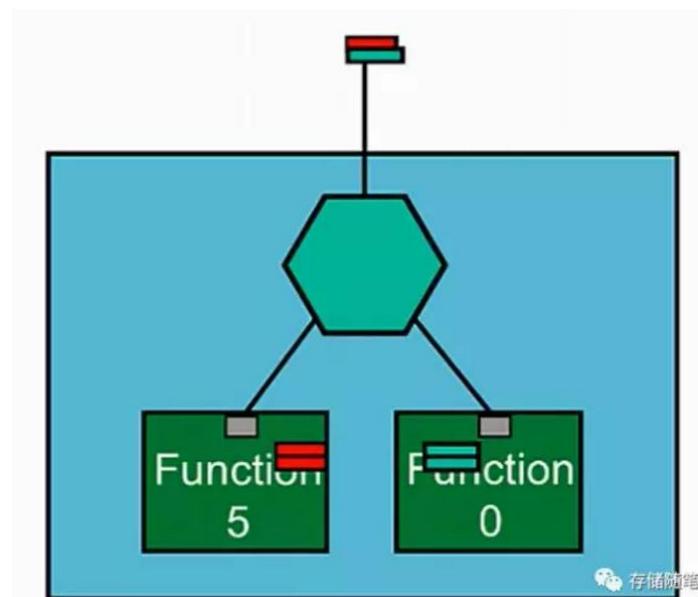
为了创造一个温馨的 FLR 工作环境，请做到以下几点：

- ✧ 在 FLR 工作期间，请不要访问对应的 Function；
- ✧ 清除所有的 Command Register；
- ✧ 通过 Polling Device Status Register 中的 Transaction Pending bit 来确保之前请求的 Completion 报文已完成，或者确保后续不会再发送 Completion 报文。
- ✧ 触发 FLR 之后，请耐心等待至少 100ms；
- ✧ 初始化 Function 的配置寄存器，让其正常工作。

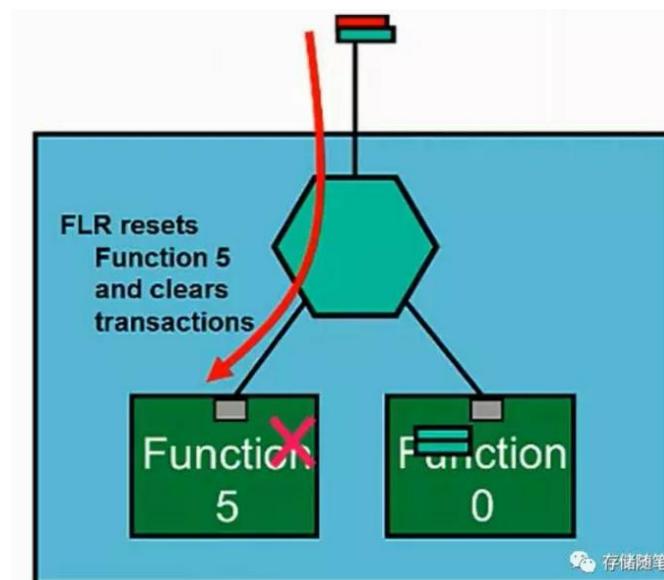
在 FLR 执行的过程中，如果收到 TLP 或者 Completions 都会被默默的丢弃，而不会向系统报错。

举个栗子，看看 FLR 执行过程，

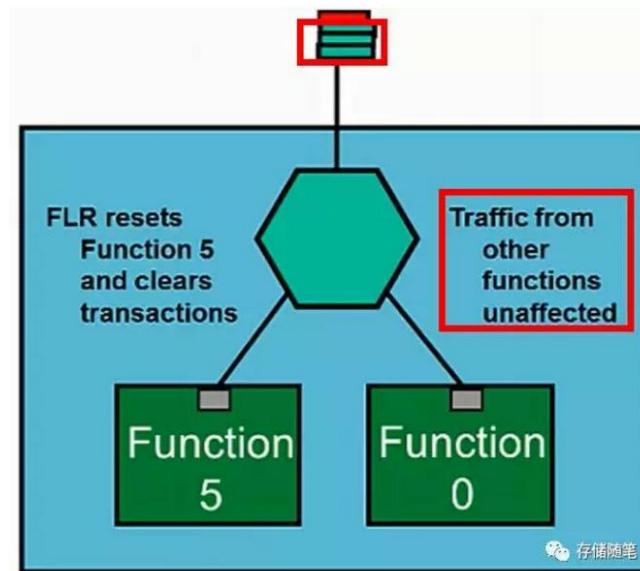
(1) 如下图，这个 PCIe 设备中有两个功能模块：Function 0 和 Function 5. 此时，两个 Functions 依旧是互不干扰，认真工作，传输 TLPs.



(2)之后，Function 5 出了一些问题，需要做 FLR。FLR 之后，Function5 中的之前的 TLPs 全部被清除。



(3) Function 5 做 FLR，并不影响 Function 0，继续 TLPs 传输。如下图，3 个 TLPs 正常传输完毕。

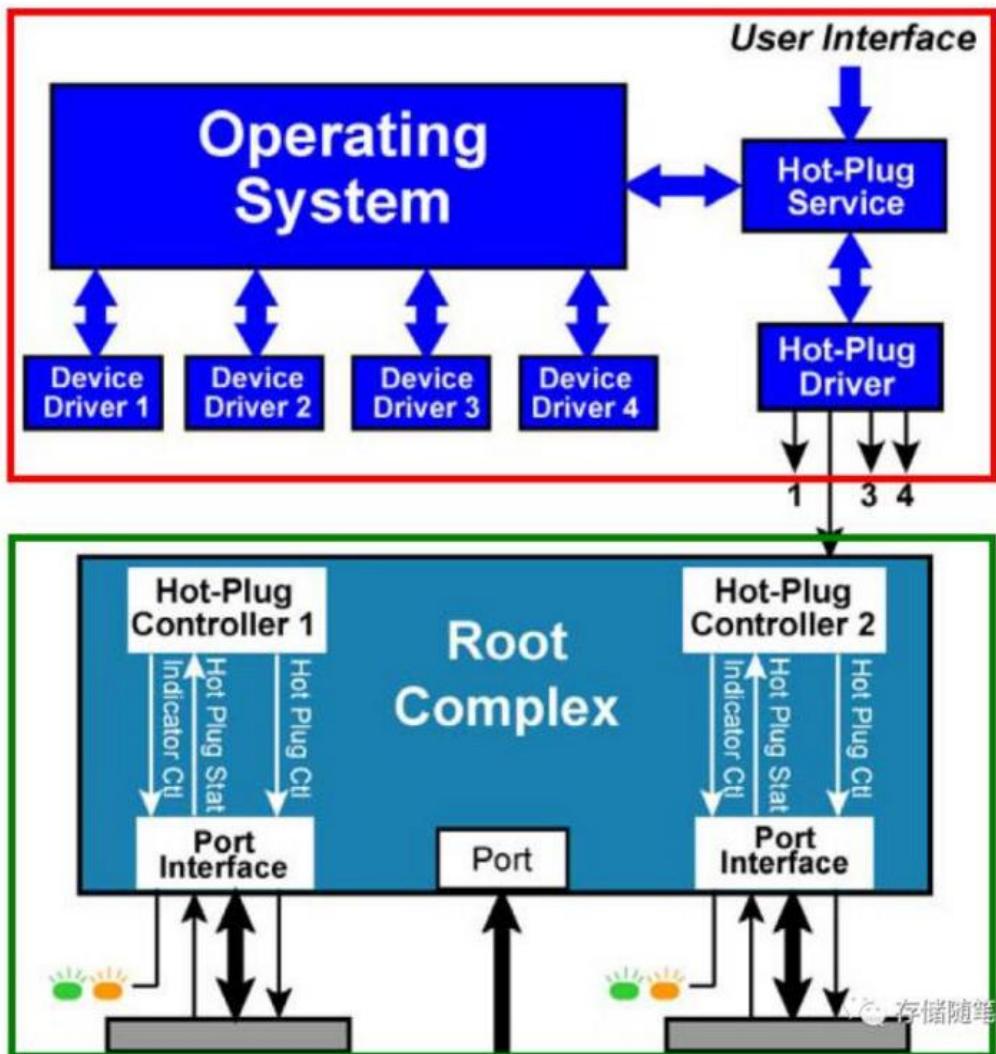


7.0 PCIe 热插拔

当你准备在不断电的情况下插拔一块 PCIe SSD 时，你要小心了。如果在 PCIe 设备不支持热插拔的条件下，很可能会对主板或 PCIe 插槽造成损毁。

为了放置意外的发生，PCIe Spec 设计了一种“No Surprise”热插拔机制，即，当用户要插拔 PCIe 设备时，必须先通知系统软件做好准备，然后通过指示灯告知用户热插拔的状态。

PCIe 环境下的热插拔需要软件与硬件的通力合作。先来看一张示意图，红色框内属于软件方面的需求，绿色框内是硬件方面的需求。



软件方面主要包括：

- ✧ **User Interface**: 这部分由系统 OS 提供。主要允许用户可以请求插拔 PCIe 设备。
- ✧ **Hot-Plug Service**: 这部分也是由系统 OS 提供。主要负责处理用户插拔 PCIe 设备的请求。
- ✧ **Standardized Hot Plug System Driver**: 这部分驱动可以由系统 OS 或者主板提供。
- ✧ **Device Driver**: 这部分主要有适配卡提供。

硬件方面主要包括：

- ✧ **Hot-Plug Controller**: 主要负责接收和处理来自 Hot Plug System Driver 的指令。
- ✧ **Card Slot Power Switching Logic**: 主要被 Hot Plug Controller 控制，用于 turn-on/off 电源。
- ✧ **Card Reset Logic**: 按照 Hot-Plug System Driver 的指示，Hot Plug Controller 向需要插拔 PCIe 设备的插槽(Slot)传送 PERST#信号。



- ✧ **Power Indicator:** 主要负责指示设备连接器上面的电源状态。
- ✧ **Attention Indicator:** 这个是警示灯，提醒用户热插拔失败状态，所以一般情况下处于关闭状态。
- ✧ **Card Present Detect Pins:** PCIe 设计了两个用于检测 PCIe 设备是否存在 的信号 PRSNT1#和 PRSNT2#。 PRSNT#1 接地，当 PCIe 设备存在时，PRSNT#2 拉高。

在介绍 PCIe 设备插拔的过程前，我们先了解一下 PCIe 插槽的 On/Off 状态：

PCIe Slot ON:

- ✧ 上电；
- ✧ RefClk 参考时钟打开；
- ✧ PCIe 链路是激活状态或者处于 ASPM 状态(L0s/L1)；
- ✧ PERST#信号处于无效状态。
- ✧ **PCIe Slot OFF:**
- ✧ 断电；
- ✧ RefClk 参考时钟关闭；
- ✧ PCIe 链路是关闭状态或；
- ✧ PERST#信号处于有效状态。

如果要调整 Slot 上的状态，步骤如下：

PCIe Slot ON 转为 OFF:

- ✧ 先关停 PCIe 链路。主要发送 EIOS 序列进入高阻态；
- ✧ 其次，向 slot 发送 PERST#信号；
- ✧ 然后，关掉 RefClk 参考时钟；
- ✧ 最后，给 slot 断电。
- ✧ **PCIe Slot OFF 转为 ON:**
- ✧ 先上电；
- ✧ 其次，打开 RefClk 参考时钟；
- ✧ 然后，解除 slot 上 PERST#信号。

在 PCIe slot 上面插拔 PCIe 设备的步骤是什么呢？

移除 PCIe 设备：

初始状态是：Attention Indicator(Yellow)-Off，Power Indicator(Green)-On

步骤是：

- ✧ 用户通过压下 Attention 按钮或者在软件界面告知系统移除 PCIe 设备的消息。当按下 Attention 按钮之后，Hot-Plug Controller 检查到这个讯息之后，会发送中断给 Root Complex。之后，Hot-Plug Service 会调用 Hot-Plug System Driver 去读取 slot 的状态信息并且侦测到 Attention 按钮的状态；
- ✧ Hot-Plug Service 调用 Hot-Plug System Driver 让 Power 指示灯开始闪烁 5s 并通过状态寄存器来验证热插拔的请求；
- ✧ Hot-Plug Service 命令 Device Driver 停用 PCIe 设备；
- ✧ 软件通过 Link Control Register 关闭 PCIe 链路；
- ✧ 软件命令 Hot-Plug Controller 关闭 slot；
- ✧ 断电后，Power 指示灯处于 OFF 状态；



- ✧ 系统为 PCIe 设备寻找对应的驱动, 并将驱动放入内存;
- ✧ 系统取消对 Slot 的配置资源。

插入 PCIe 设备:

初始状态是: Attention Indicator (Yellow)-Off, Power Indicator (Green)-Off

步骤是:

- ✧ 用户安装 PCIe 设备, 并且压下 Attention 按钮或者在软件界面告知系统安装 PCIe 设备的信息。主要通过发送中断的形式告知系统热插拔信息;
- ✧ 热插拔软件通过状态寄存器来验证热插拔的请求;
- ✧ 软件命令 Power 指示灯开始闪烁;
- ✧ 软件命令 Hot-Plug Controller 将 Slot 打开, 让 Slot 处于 ON 状态;
- ✧ 上电后, Power 指示灯处于 ON 状态;
- ✧ 系统为 PCIe 设备寻找对应的驱动, 并将驱动放入内存;
- ✧ 系统调用驱动完成对 PCIe 设备的初始化。

更多精彩内容, 敬请关注微信公众号: 存储随笔, Memory-logger.



声明: 本教程版权归微信公众号"存储随笔"所有, 支持原创, 转载请申明出处!