

# 灵衢<sup>®</sup>基础规范

文档版本：2.0

发布日期：2025年9月

**版权所有 © 2025 华为技术有限公司。保留一切权利。**

本规范的版权由华为技术有限公司（以下简称“华为”）和/或其受让人所有。您对本规范的使用受《灵衢规范许可协议 V1.0》（“本协议”）中规定的条款和条件的约束。如果您不同意本协议，请勿下载、持有或使用本规范。

华为根据本协议授予您全球范围内的、非排他的、非独占的、不可分许可的、不可转让的、免费的版权许可，该版权许可仅允许您为开发符合产品的目的（以下简称“协议目的”）实施灵衢规范。

您同意，不得出于非协议目的以任何形式实施灵衢规范，也不得以任何方式修订、更改、修改灵衢规范或制作衍生作品，例如，不得摘录或引用灵衢规范的任何内容用于开发其他标准。

本协议中没有授予除明示规定外的任何其他权利，包括但不限于未授予您使用华为商标或服务标志的授权。本协议未明示授予的任何其他权利均由华为保留。您需要在如下网址查看本协议全文：<https://www.unifiedbus.com/zh/specification-license-agreement-v1>。

华为可能在其认为必要或适当的情况下对本规范进行更新，您同意华为对本规范的更新不需要事先对您进行通知。

除非适用法强制规定或者双方有明确书面约定，华为对本规范中的所有陈述、信息和建议不做任何明示或默示的担保或保证，包括但不限于不侵权、无错误、时效性或满足特定目的的保证。

华为对因您使用本规范而造成的损害均不承担任何责任，包括但不限于因本规范中的任何缺陷、错误或遗漏或因侵犯任何第三方的任何知识产权而导致的损害。

灵衢<sup>®</sup>和 UnifiedBus<sup>™</sup>均为华为商标。本规范中提及或展示的其它商标、产品名称、服务名称以及公司名称，由各自的所有人拥有。

# 目录

<b>1 简介.....</b>	<b>8</b>
1.1 目的.....	8
1.2 范围.....	8
1.3 结构.....	8
1.4 规范性引用文件.....	8
1.5 术语定义.....	9
<b>2 架构.....</b>	<b>13</b>
2.1 概述.....	13
2.2 协议栈.....	15
<b>3 物理层.....</b>	<b>17</b>
3.1 概述.....	17
3.2 物理编码子层.....	19
3.3 物理媒介适配子层.....	45
3.4 链路状态管理.....	47
<b>4 数据链路层.....</b>	<b>87</b>
4.1 概述.....	87
4.2 数据链路状态机.....	87
4.3 DLLCB 和 DLLDP 收发.....	89
4.4 初始化自协商.....	119
4.5 虚拟通道机制.....	122
4.6 信用流控机制.....	122
4.7 误码检测和重传机制.....	125
4.8 异常处理.....	134
<b>5 网络层.....</b>	<b>136</b>
5.1 概述.....	136
5.2 网络层包头（NTH）.....	137
5.3 网络层特性.....	140

<b>6 传输层 .....</b>	<b>149</b>
6.1 概述 .....	149
6.2 传输层 packet 格式 .....	150
6.3 传输层模式 .....	159
6.4 RTP 可靠传输机制 .....	160
6.5 多路径负载均衡 .....	184
6.6 拥塞控制机制 .....	187
6.7 传输流程 .....	191
6.8 传输层事务层交互流程 .....	194
<b>7 事务层 .....</b>	<b>197</b>
7.1 概述 .....	197
7.2 事务层包头 ( TAH ) .....	198
7.3 事务服务 .....	213
7.4 事务类型 .....	222
<b>8 功能层 .....</b>	<b>240</b>
8.1 概述 .....	240
8.2 基本概念 .....	240
8.3 Load/Store 同步访问 .....	252
8.4 URMA 异步访问 .....	253
8.5 URPC .....	254
8.6 多 Entity 协同 .....	257
8.7 Entity 管理 .....	258
<b>9 内存管理 .....</b>	<b>259</b>
9.1 概述 .....	259
9.2 Home-User 访问模型 .....	259
9.3 UB 内存描述符 .....	260
9.4 UMMU 功能与处理流程 .....	260
9.5 UB Decoder 功能与流程 .....	285
<b>10 资源管理 .....</b>	<b>288</b>

## 目录

10.1 概述.....	288
10.2 基本概念.....	288
10.3 工作机制.....	291
10.4 管理机制.....	301
10.5 虚拟化.....	333
10.6 RAS .....	335
<b>11 安全.....</b>	<b>343</b>
11.1 概述.....	343
11.2 设备认证.....	345
11.3 资源分区隔离.....	347
11.4 访问控制.....	347
11.5 数据通路保护.....	350
11.6 可信执行环境扩展 .....	354
<b>附录 A 缩略语.....</b>	<b>362</b>
<b>附录 B 包格式.....</b>	<b>369</b>
B.1 概述 .....	369
B.2 包格式.....	370
B.3 UPI Header (UPIH) .....	374
B.4 EID Header (EIDH) .....	375
<b>附录 C GUID 和 Class Code 编码 .....</b>	<b>376</b>
C.1 GUID 编码定义 .....	376
C.2 Class Code 编码定义 .....	377
C.3 编码使用 .....	377
<b>附录 D 配置空间寄存器 .....</b>	<b>379</b>
D.1 CFG0_BASIC.....	379
D.2 CFG0_CAP .....	385
D.3 CFG1_BASIC.....	405
D.4 CFG1_CAP .....	413
D.5 CFG0_PORT_BASIC.....	422

## 目录

D.6 CFG0_PORT_CAP .....	424
D.7 CFG0_ROUTE_TABLE.....	498
<b>附录 E 以太互通 .....</b>	<b>501</b>
<b>附录 F 基于 UB 链路的网络管理 .....</b>	<b>502</b>
F.1 适用场景.....	502
F.2 管理协议.....	502
<b>附录 G 设备热插拔 .....</b>	<b>508</b>
G.1 一般要求.....	508
G.2 热插拔组件 .....	508
G.3 热拔出流程 .....	510
G.4 热插入流程 .....	510
G.5 热插拔事件 .....	511
<b>附录 H URPC 消息格式.....</b>	<b>512</b>
H.1 概述 .....	512
H.2 URPC Function .....	512
H.3 URPC Message .....	513
<b>附录 I 应用示例.....</b>	<b>518</b>
I.1 存储 PLOG .....	518

## 修订历史

版本	时间	变化
2.0	2025-09	初始发布

# 1 简介

## 1.1 目的

本文档定义灵衢基础规范 2.0 版本，包括灵衢系统组成、协议、以及编程模型等。本文档帮助读者理解符合灵衢基础规范的设备和系统的交互行为、互操作要求、编程模型、资源管理等。

本文档读者对象包括但不限于如下群体：

- 芯片/IP 设计、开发、测试工程师、验证工程师
- 软件设计与开发工程师
- 硬件设计与开发工程师
- 标准工程师
- 产品经理专家

## 1.2 范围

本文档包括架构、物理层、数据链路层、网络层、传输层、事务层、功能层等内容。

## 1.3 结构

灵衢规范系列由基础规范和一组配套的规范组成。基础规范包括灵衢系统组成、协议栈、编程模型等技术细节。配套规范包括灵衢固件规范、灵衢使能操作系统参考设计等。

## 1.4 规范性引用文件

1. IANA 地址解析协议 ( ARP ) 参数 ( Address Resolution Protocol (ARP) Parameters )
2. IANA 服务名称与传输协议端口号注册表 ( Service Name and Transport Protocol Port Number Registry )
3. IEEE Std 802.3™ - 2022 IEEE 以太网标准 ( IEEE Standard for Ethernet )
4. IEEE Std 802.3ck™ - 2022 IEEE 以太网每通道 100Gb/s 物理层标准 ( IEEE Standard for Ethernet Amendment 4: Physical Layer Specifications and Management Parameters for 100 Gb/s, 200 Gb/s, and 400 Gb/s Electrical Interfaces Based on 100 Gb/s Signaling )
5. IEEE 802.1AX-2020 链路聚合 ( IEEE Standard for Local and Metropolitan Area Networks--Link Aggregation )
6. IETF RFC 768 用户数据报协议 ( User Datagram Protocol (UDP) )

7. IETF RFC 791 互联网协议 ( Internet Protocol )
8. IETF RFC 2460 互联网协议版本 6 ( Internet Protocol, Version 6 (IPv6) Specification )
9. IETF RFC 2131 动态主机配置协议 (Dynamic Host Configuration Protocol)
10. IETF RFC 8415 IPv6 动态主机配置协议 (Dynamic Host Configuration Protocol for IPv6)
11. DMTF DSP0236 管理组件传输协议 ( MCTP ) 基础规范 ( Management Component Transport Protocol (MCTP) Base Specification )
12. DMTF DSP0274 安全协议与数据模型( SPDM )规范( Security Protocol and Data Model (SPDM) Specification )
13. GM/T 0002-2012 SM4 分组密码算法
14. NIST SP 800-38D 推荐使用分组密码操作模式：伽罗瓦/计数器模式 ( GCM ) 和 GMAC ( Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC )
15. OIF CEI-112G-LINEAR-PAM4 112G 线性光学组件电气标准

## 1.5 术语定义

术语	中文名称	定义
Cell	信元	信用证的基本单元，用于信用流控。
Compact Network Address ( CNA )	简短网络地址	支持 16 bit 和 24 bit 两种格式的网络地址。
Compact Transport ( CTP )	轻量级传输模式	一种传输层模式，借助下层协议共同提供可靠和拥塞控制的传输服务。
Configuration Space	配置空间	用于存放设备的能力、状态、配置等信息的设备存储区域，为软件提供 Entity 配置管理接口。
Data Link Layer Data Block ( DLLDB )	数据链路层数据块	组成数据链路层数据包的数据单位，长度为 1~32 个 Flits。
Entity	实体	设备分配其自身资源的基本单元，每个 Entity 都是 UB 域内的一个通信对象。
Entity Identifier ( EID )	Entity 身份标识	分配给 Entity 使用，唯一标识 Entity 的通信对象身份。
Execution Environment bits ( EE_bits )	执行环境标识位	标识可信执行环境安全通信请求来源所处计算环境的安全状态。
Flit	微片	组成数据链路层包的基本单位，长度为 20 个字节。
Globally Unique Identifier ( GUID )	全局唯一身份标识	Entity 的永久身份标识，具备全局唯一性。

术语	中文名称	定义
Home	待访问方	Home–User 访问模型中内存的拥有者。
Initiator	发起端	发起事务请求的 Entity。
Jetty	通信码头	URMA 基本通信单元，提供异步事务操作下发和执行能力，支持多对多和 1 对 1 等通信模式。
Memory Segment	内存段	一段连续地址的内存，是内存事务的基本操作对象，由全局唯一的 UBMD 标识。
Maximum Transmission Unit ( MTU )	传输层最大传输单元	UB 传输层 packet 能够承载的最大事务层净荷大小
Network Partition	网络分区	一组 IP 网络接口设备的集合，不同网络分区之间的网络通信隔离。
Reliable and Ordered by Initiator ( ROI )	可靠和发送端保序模式	可靠的、Initiator 完成事务保序的事务服务模式，该模式下事务层包可乱序传输。
Reliable and Ordered by Lower Layer ( ROL )	可靠和下层保序模式	可靠的、由下层协议完成事务保序的事务服务模式。该模式下事务层包是否可乱序传输根据下层协议确定。
Reliable and Ordered by Target ( ROT )	可靠和目的端保序模式	可靠的、Target 完成事务保序的事务服务模式，该模式下事务层包可乱序传输。
Reliable Transport ( RTP )	可靠传输模式	一种传输层模式，提供端到端可靠无冗余传输服务。
Resource Space	资源空间	用于存放中断信息、设备功能相关的配置信息和厂商自定义信息的设备存储区域，为软件操作提供 Entity 配置管理接口。
RX Lane	接收通道	端口的接收通道，一个端口的标准接收通道数是 1, 2, 4, 8。UB 支持非对称链路，一个端口的接收通道数量可以和发送通道数量不一样。
Target	目的端	接收并处理事务请求的 Entity。
Token	令牌	用于鉴别请求发起者能否访问目标内存或 Jetty 的凭据，包括标识符 TokenID 或代表接收队列的索引和凭据值 TokenValue。
Transaction Complete Order ( TCO )	事务完成序	一组事务在执行完成后上报事务完成的顺序，包含发送完成序和接收完成序两种类型。
Transaction Execute Order ( TEO )	事务执行序	一组事务在 Target 的执行顺序。

术语	中文名称	定义
Transport Acknowledgement ( TPACK )	传输正确应答	Transport Receiver 返回给 Transport Sender 的正确应答信号，以指示成功接收的 Transport Packet，用于保证端到端传输的可靠性。
Transport Channel Group ( TPG )	传输通道组	多个传输通道的组合，实现流量在多个传输通道间的负载均衡。
Transport Channel ( TP Channel )	传输通道	在两个传输端点间建立的端到端的连接，为事务层提供端到端可靠通信。
Transport Endpoint ( TPEP )	传输端点	参与传输层通信的端点，用于发送和接收各种传输层包。
Transport Negative Acknowledgement ( TPNAK )	传输错误应答	传输接收方返回给传输发送方的错误应答信号，以提供明确的错误信息，用于保证端到端传输的可靠性。
Transport Receiver ( TP Receiver )	传输接收方	接收传输层数据包的传输端点。
Transport Sender ( TP Sender )	传输发送方	发送传输层数据包的传输端点。
Trusted Execution Environment ( TEE )	可信执行环境	基于硬件级隔离及安全启动机制，为确保安全敏感应用相关数据和代码的机密性、完整性、真实性和不可否认性目标构建的一种计算环境。[来源：GB/T 41388-2022 ,3.3,有修改]
TX Lane	发送通道	端口的发送通道，一个端口的标准发送通道数是 1, 2, 4, 8。UB 支持非对称链路，一个端口的发送通道数量可以和接收通道数量不一样。
UB Address ( UBA )	UB 地址	待访问方提供给访问发起方的地址，用于访问待访问方的内存段。
UB Decoder	UB 译码器	负责将访问方物理地址转换为 UB 地址的硬件。
UB Domain	UB 域	一个全部使用 UB Link 连接起来的 UBPU 集合。
UB Fabric	UB 互连结构	UB Domain 内所有 UB Switch 和 UB Link 的集合。
UB Link	UB 链路	两个端口以及他们之间相连的若干条 TX Lane 和若干条 RX Lane 的集合。UB link 的 TX Lane 数量和 RX Lane 的数量可以不一样。
UB Memory Descriptor ( UBMD )	UB 内存描述符	UB 内存描述符包括 EID、TokenID 和 UBA，用于索引待访问方的物理地址。

术语	中文名称	定义
UB Memory Management Unit ( UMMU )	UB 内存管理单元	将 UB 内存描述符转换为待访问方的物理地址并进行权限校验的组件。
UB over Ethernet ( UBoE )	以太网承载 UB	一种通过以太/IP 网络传输 UB 事务的方式，其包可以被 IP 网络路由。
UB Partition	UB 分区	一组 Entity 的集合，不同集合之间 UB 事务通信隔离。
UB Processing Unit ( UBPU )	UB 处理单元	支持 UB 协议栈的处理单元，实现特定功能。
Unified Remote Memory Access ( URMA )	统一远程内存访问	一种支持异步访问编程的高速通信库，提供异步内存访问和双边消息通信功能。
Unified Remote Procedure Call ( URPC )	统一远程过程调用	一套基于 UB 事务层能力提供的函数访问协议，允许 UBPU 之间直接进行远程平等函数调用。
UnifiedBus ( UB )	灵衢	一种面向超节点的互联协议，简称 UB。
Unreliable and Non-Ordering ( UNO )	不可靠无保序模式	不可靠的、无事务序事务服务模式，该模式下事务层包可乱序传输。
Unreliable Transport ( UTP )	不可靠传输模式	一种传输层模式，提供不可靠的传输服务。
User	访问方	Home–User 访问模型中的内存访问者。

# 2 架构

## 2.1 概述

灵衢 (UnifiedBus, UB) 是一种面向超节点的互联协议，将 IO、内存访问和各类处理单元间的通信统一在同一互联技术体系，实现高性能数据搬移、资源统一管理、资源灵活组合、处理单元间高效协同和高效编程。

基于灵衢建立的计算系统，称之为灵衢系统，部署范围可以从单台服务器到全数据中心。基于统一互联，灵衢系统中的所有处理单元地位平等、所有资源均可池化。灵衢系统组成如图 2-1、图 2-2 所示：

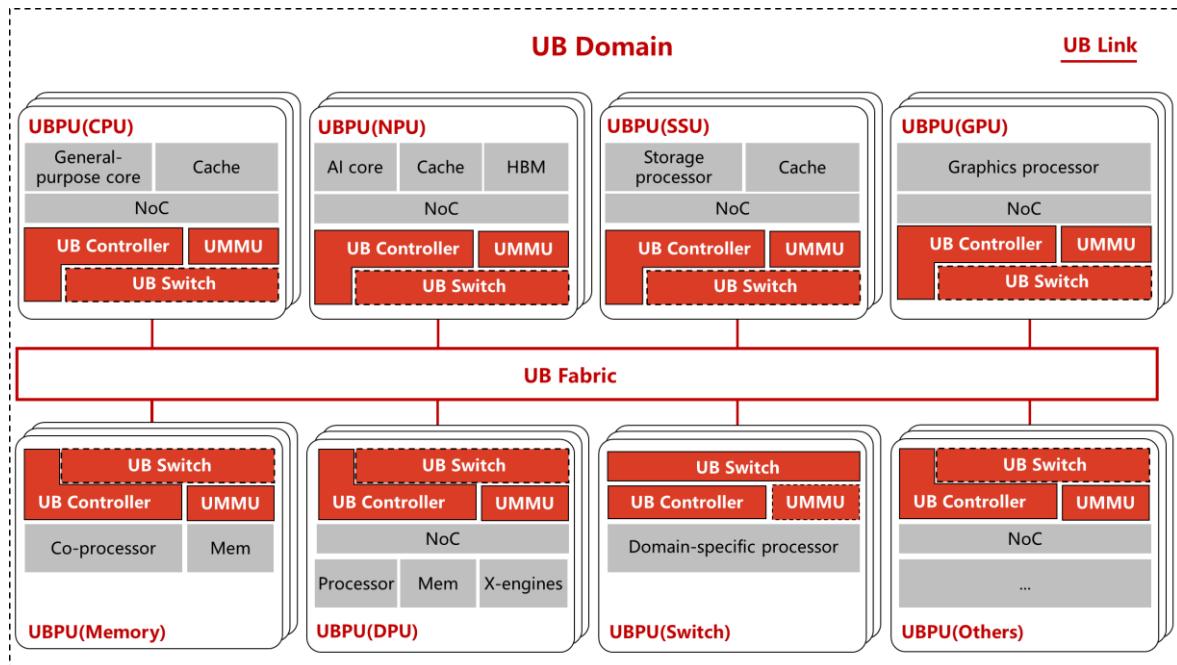


图 2-1 UB Domain 系统组成

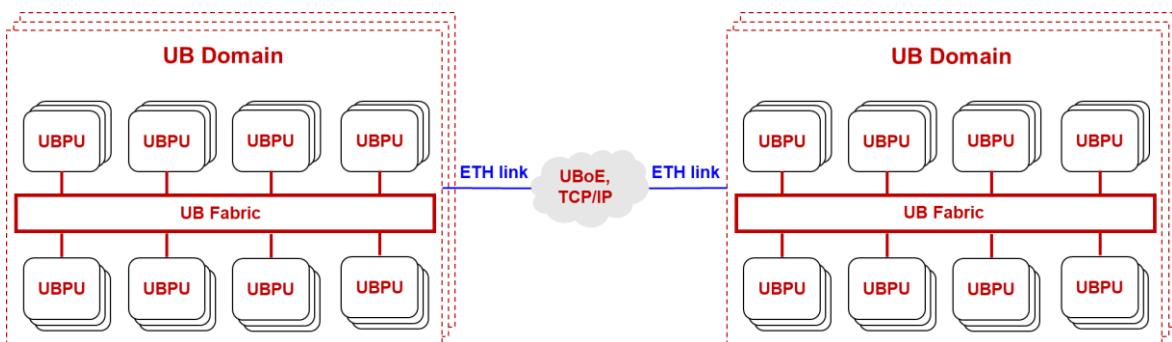


图 2-2 跨 UB Domain 系统组成

UB 包含如下要素：

- UB Processing Unit ( **UBPU** ) 是支持 UB 协议栈的处理单元，实现特定功能。
- **UB Controller** 是 UBPU 中执行 UB 协议栈的组件，并提供软硬件接口。
- UB Memory Management Unit( **UMMU** )是 UBPU 中执行内存地址映射和访问权限校验的组件。
- **UB Switch** 是 UBPU 中的可选组件，支持在 UB 端口间转发包。
- **UB Link** 是 UBPU 间的点到点连接。
- **UB Domain** 是一个全部使用 UB Link 连接起来的 UBPU 集合。
- **UB Fabric** 是 UB Domain 内所有 UB Switch 和 UB Link 的集合。
- UB over Ethernet ( **UBoE** ) 通过以太/IP 网络承载 UB 事务，实现跨 UB Domain 互通。

UB 主要特征包括：

- **协议归一**，从计算硬件系统角度，用户所有的操作归为内存访问、消息传递、过程调用、资源管理。UB 使用单一协议栈支持上述所有操作，避免协议转换开销，为软件开发者提供利用硬件能力的高效方式。
- **平等协同**，UBPU 自身处理能力各有所长，但在协议功能上地位平等。任意 UBPU 均可直接访问其它 UBPU，其访问请求和对应的数据通路无需第三方 UBPU 介入。任意 UBPU 均可调用其它 UBPU 提供的功能，实现多个 UBPU 协同完成任务。
- **全量池化**，UB Domain 内任意 UBPU 均可共享自己的计算、内存、存储资源或使用其它 UBPU 共享的资源，在满足业务的 SLA 下实现系统内资源的最大化利用。UBPU 支持以 Entity 粒度把资源划分给不同的用户使用，可按需组合特定配置的计算节点；计算资源池化允许同构计算资源弹性伸缩、异构计算资源按需组合完成特定任务；内存池化允许 UB Domain 内所有内存的借用和共享。互联池化涉及多样化的互联资源共享，如传输层通道面向多个 Entity 的共享、多端口聚合通信、端到端多路径等，可最大化互联带宽利用率。UB Domain 引入 UBFM，通过集中管理和调度让池化资源更有序和高效的使用。
- **全栈协同**，UB 协议栈基于分层设计，每层根据可靠性、时延、带宽等需求提供协议配置项，并且可按需组合兼顾性能和可靠性的多种传输模式、同步和异步内存语义，满足大带宽、低时延的规模组网需求。
- **任意拓扑**，UBPU 是计算/数据存储/交换的融合体，可综合吞吐、延迟、资源利用、能耗、可用性等多维度策略在合适的拓扑位置提供计算处理，使系统处理效率最优。UB 支持链路层的虚通道机制、网络层逐包/逐流多路径路由机制，传输层传输通道组共享机制，从而支持 nd-mesh、Clos、torus 等在内的任意拓扑或拓扑组合，以提升系统性能、增进容错性、支持大规模连接、降低部署成本。
- **高可用机制**，UB 支持物理层故障时降速或降 lane 以及故障恢复后升速或升 lane、链路层点到点重传、网络多路径、传输层端到端重传、故障隔离和就近处理、分层和系统级 Reliability, Availability, and Serviceability ( RAS ) 特性等多重技术，支撑系统高可用性。

## 2.2 协议栈

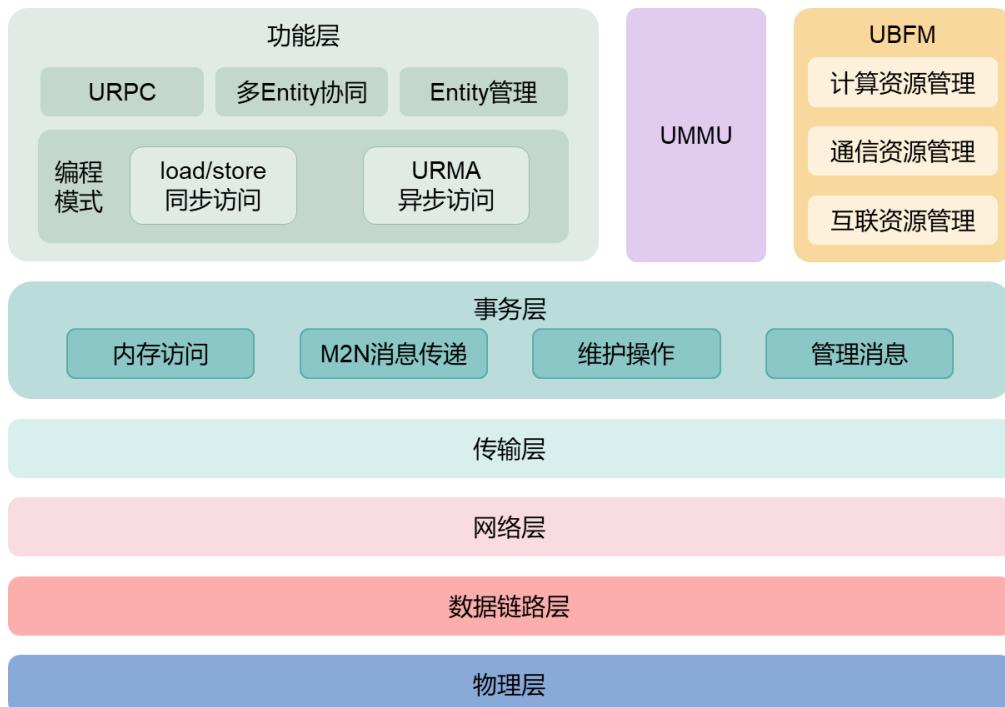


图 2-3 UB 协议栈

UB 协议简介如下：

- **物理层**，负责在物理介质上为数据链路层提供比特流传输。物理层支持自定义速率，充分利用 SerDes 及信道最大能力。物理层支持多种 FEC 模式及 FEC 模式动态切换，匹配不同链路 BER 特征，降低时延。物理层支持故障时降速或降 lane 以及故障恢复后升速或升 lane，使能光模块通路保护，增强链路可用性。
- **数据链路层**，在 UB Link 的两个端口间为网络层包提供可靠传送服务。为确保数据在点到点链路上可靠、无差错传送，链路层提供循环冗余校验（Cyclic Redundancy Check, CRC）检验、包重传、信用证流控、虚通道。
- **网络层**，为上层协议提供 UB Domain 内或跨 UB Domain 的路由服务。网络层支持标准 IP 地址格式和简短网络地址格式，满足上层协议对兼容性和效率的不同要求。网络层支持多路径通信能力，提供逐包和逐流两种负载均衡模式，支持由上层协议为业务流指定选路策略，最大化多路径网络的带宽利用率。
- **传输层**，为事务层提供多种可选择的端到端传输服务。可靠传输模式（Reliable Transport, RTP）提供面向连接和可靠无冗余传输服务，一般用于端到端无损传输；轻量级传输模式（Compact Transport, CTP）借助下层协议如链路层提供传输的可靠性，可应用于端到端路径故障低的通信环境；不可靠传输模式（Unreliable Transport, UTP）提供无连接和不可靠传输的传输服务，一般用于对丢包不敏感的业务如带内建立连接等业务。传输层支持多个 TP Channel 统一调度，实现流量在多 TP Channel 间的负载均衡。传输层还提供负载均衡和拥塞管理机制，降低动态传输

时延，实现包的高效传输。此外，传输层可配置为旁路模式（Transport Bypass）支持事务层直接调用网络层服务，可降低协议开销。

- **事务层**，为上层应用（含 UB 协议定义的功能）提供内存操作、消息操作、维护操作以及管理操作四种事务操作，实现同步内存访问、异步内存访问、消息传递、状态维护、Entity 管理等应用。统一事务操作，屏蔽不同编程模型下事务交互流程的差异，提供可靠和发送端保序、可靠和目的端保序、可靠和下层保序、不可靠无保序四种事务服务模式。
- **功能层**，提供 Load/Store 同步访问和统一远程内存访问（Unified Remote Memory Access，URMA）异步访问两类编程模型。对于 Load/Store 同步访问，UB Controller 配合片上总线将 Load/Store 指令转换为事务操作（如 read、write、atomic 等）；对于 URMA 异步访问，应用可调用 Jetty 提供的接口，完成通信对关系建立、事务操作提交与查询响应等功能。在编程模型基础上，还允许进一步的编程抽象，如统一远程过程调用（Unified Remote Procedure Call，URPC）提供基于内存对象的函数调用的描述，可实现任意 UBPU 间的功能调用。
- **UBFM**，UBFM 是 UB Domain 的管理者，负责系统内的计算资源管理、互连资源管理和通信管理。根据 UB Domain 的规模，可部署多 UBFM 实例协同实现对 UB Domain 的管理。
- **UMMU**，UMMU 在内存访问过程中提供内存地址映射和访问权限校验，可支持 UBPU 间内存资源的共享，以及确保对内存的合法访问。
- **安全**，UB 允许数据中心范围的资源池化和组合，并支持多租户，需要确保资源访问安全。UB 提供设备身份认证、资源访问隔离、访问控制、数据通路传输机密性和完整性保护，和跨设备的可信执行环境扩展等安全特性，允许安全可信的共享和组合资源，保护内存数据使用安全、总线数据传输安全。

# 3 物理层

## 3.1 概述

物理层向上为数据链路层提供数据收发服务；向下通过 SerDes ( Serializer/Deserializer ) 串行通道与对端 UBPU 连接。物理层包含物理编码子层，物理媒介适配子层和链路状态管理功能，如下图所示：

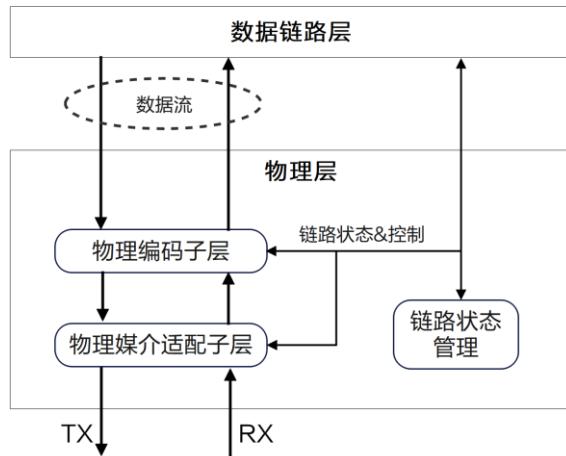


图 3-1 物理层概览

物理编码子层 ( Physical Coding Sublayer, PCS ) 完成数据的 FEC ( Forward Error Correction ) 编解码和扰码。在发送方向上 PCS 从数据链路层接收数据，对其进行 FEC 编码和加扰后发送给 PMA；在接收方向上 PCS 从 PMA 接收数据，完成解扰和 FEC 译码后将数据发送到数据链路层。

物理媒介适配子层 ( Physical Medium Attachment, PMA ) 从串行通道上收发比特流，完成时钟恢复、信号调制解调、格雷编码、预编码、并串转换等功能，支持物理编码子层通过介质无关方式与多种物理链路连接。

链路状态管理通过链路管理状态机实现链路训练及链路故障恢复功能，完成链路两端的速率、链路宽度、均衡参数及编码模式等的协商。成功完成链路训练后，物理层可以向数据链路层提供数据收发服务。

UB 物理层具有如下特征：

- 支持自定义速率，不受限于固定的标准速率，充分利用 SerDes 及信道能力。
- 支持多种 FEC 模式及 FEC 模式动态切换，匹配不同的链路误码率 ( Bit Error Ratio, BER ) 特征，降低时延。
- 支持故障时降低通道或者速率，支持光模块通路保护，增强链路可用性。
- 支持不对称上下行链路带宽，节省功耗。

### 3.1.1 端口、通道与链路介绍

端口 ( Port ) 由发送器 ( TX ) 和接收器 ( RX ) 组成，分别连接到对端端口的接收器和发送器。每个 TX 或者 RX 包括若干条通道 ( Lane )，每条通道是一对高速差分信号线。UBPU 的一个端口和对端 UBPU 的一个端口通过双向若干条通道连接起来组成一条链路 ( Link )。UB 支持电链路，也支持光链路。

TX 或者 RX 包括的通道数可以是 1、2、4 或 8。UB 支持对称链路，即一个端口的 TX 和 RX 的通道数相同；也支持不对称链路，即一个端口的 TX 和 RX 的通道数不同。如下图所示，从 PortA 的角度来看，TX 包括 M 条通道，RX 包括 N 条通道。M 可以等于 N，也可以不等于 N。

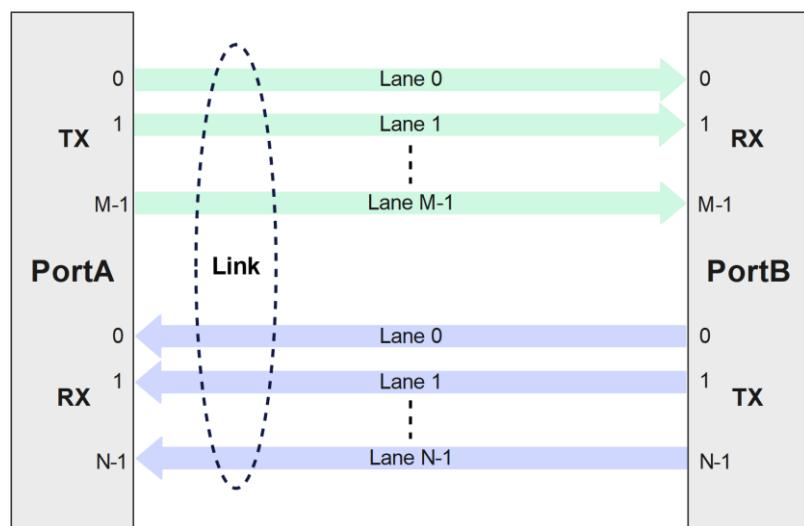


图 3-2 UB 链路组成示意

UB 的不对称链路支持静态配置，也支持在不中断业务时动态改变任一方向的链路宽度（即链路所包括的通道数）。

### 3.1.2 物理层模式

物理层支持两种模式，如下表所示：

表 3-1 UB 物理层模式

模式	PHY Mode-1	PHY Mode-2
数据速率 ( Gbps )	4.0, 自定义速率	2.578125, 25.78125, 53.125, 106.25
调制模式	4.0 Gbps: NRZ 自定义速率: NRZ 或 PAM4	2.578125/25.78125 Gbps: NRZ 53.125/106.25 Gbps: PAM4

Port 在系统上电前应配置成某一种物理层模式，系统上电后不能动态切换。

UB 物理层支持的电气特征如下：

- 在 DAC 电缆、背板、有重定时的光模块场景，PHY Mode-2 中数据速率的电气特征可参考 IEEE Std 802.3™-2022 及 IEEE Std 802.3ck™-2022。
- 在无重定时的线性光组件场景，106.25 Gbps 速率的电气特征可参考 OIF CEI-112G-LINEAR-PAM4。
- UB 可选的在背板场景支持最高速率 118Gbps，最大插损 40dB(bump2bump,@奈奎斯特频率)。
- UB 可选的在 DAC 电缆场景支持最大插损 42dB(bump2bump,@奈奎斯特频率)。

支持自定义速率的 Port 在产品出厂前应设定好具体速率值以及调制模式，使用时应确保链路两端支持的此自定义速率值和调制模式相同。

## 3.2 物理编码子层

### 3.2.1 数据通路概述

物理层数据通路包括 PCS 和 PMA。发送方向，PCS 从数据链路层接收数据，进行 FEC 编码（FEC 旁路可选）、加扰后，通过 PMA 的格雷编码（针对 PAM4 调制需要格雷编码）、预编码及信号调制后发送到链路上。帧定界与控制码字（Alignment Marker Control，AMCTL，见 3.2.4 节）被定期插入数据流（链路正常工作时）或链路训练码流（链路训练期间）之间，完成帧定界和执行各种控制功能。接收方向执行反向动作。物理层数据通路功能如下图所示：

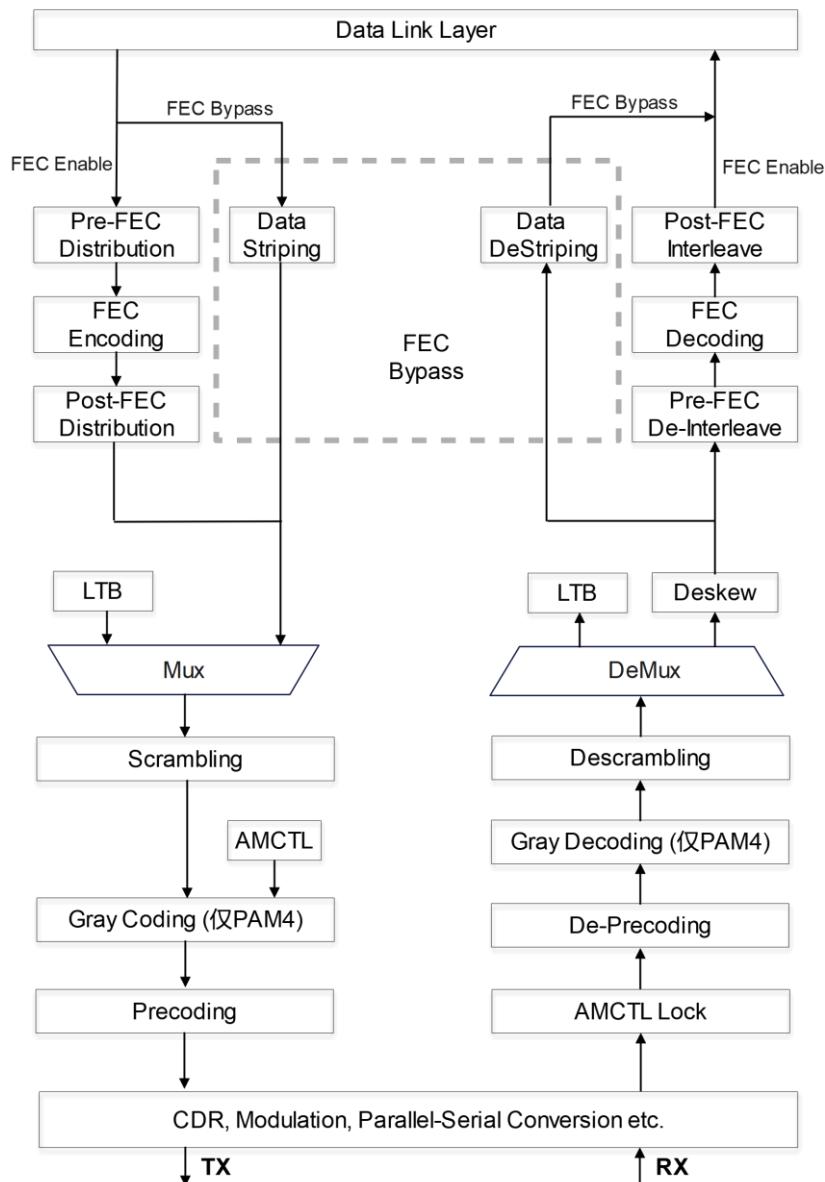


图 3-3 物理层数据通路功能图

在 FEC 旁路 ( FEC Bypass ) 时，数据不进行 FEC 编解码，在发送方向数据以 symbol 为单位顺序发送到各条 Lane 上 ( Data striping )，接收方向执行反向动作 ( Data destriping )。

### 3.2.2 发送侧编码子层功能

#### 3.2.2.1 前向纠错编码

PCS 使用 RS ( Reed-Solomon ) 码字进行编码。RS 码字工作在有限域空间。编码器对接收到的 K 个 FEC 消息符号进行编码，生成 N-K 个校验符号，组成 N 个 RS FEC 符号，得到 RS(N,K,T) 的 RS FEC 码字。

RS FEC 编码器工作在有限域  $GF(2^m)$ ，其中 m 为每个 RS FEC 符号中含有的比特数。

RS FEC 码字的生成多项式  $g(x)$  如下所示：

$$g(x) = \prod_{j=0}^{2T-1} (x - \alpha^j) = g_{2T}x^{2T} + g_{2T-1}x^{2T-1} + \dots + g_1x + g_0$$

其中， $\alpha$  为  $GF(2^m)$  的本原多项式的根。对于  $GF(2^8)$  有限域，其本原多项式为  $\alpha=x^8+x^4+x^3+x^2+1$ 。

RS FEC 的消息多项式用  $m(x)$  表示，其表达式如下：

$$m(x) = m_{K-1}x^{N-1} + m_{K-2}x^{N-2} + \dots + m_1x^{2T+1} + m_0x^{2T}$$

其中第  $i$  个消息符号  $m_i = m_{i,7}\alpha^7 + m_{i,6}\alpha^6 + \dots + m_{i,1}\alpha + m_{i,0}$ ，消息符号输入到 RS FEC 编码的顺序为  $m_{K-1}, m_{K-2}, \dots, m_1, m_0$ 。

RS FEC 校验多项式  $p(x)$  的表达式如下，系数由  $p_{2T-1}, p_{2T-2}, \dots, p_0$  组成：

$$p(x) = p_{2T-1}x^{2T-1} + p_{2T-2}x^{2T-2} + \dots + p_1x + p_0$$

校验多项式由消息多项式  $m(x)$  除以生成多项式  $g(x)$  取余得到。

RS FEC 编码器电路如下图所示：

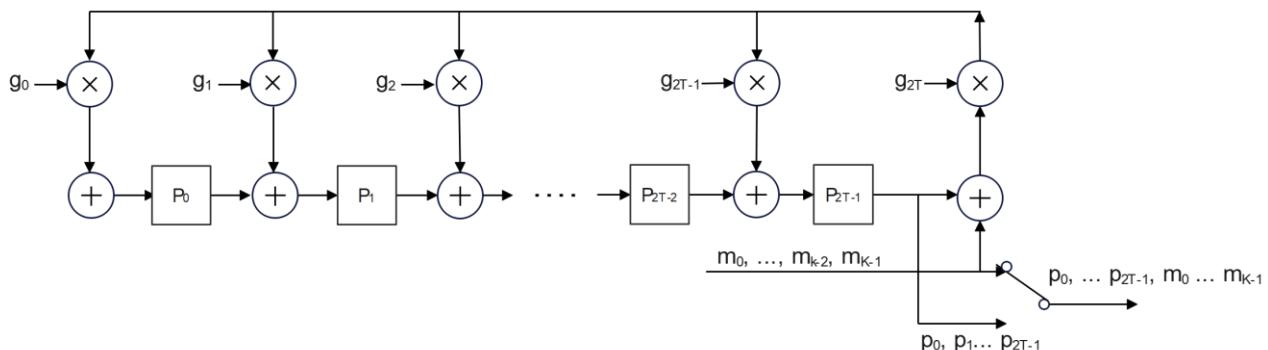


图 3-4 RS FEC 编码电路

RS FEC 编码后的符号从先到后依次是： $m_{k-1}, m_{k-2}, \dots, m_0, p_{2T-1}, \dots, p_0$ 。

FEC 编码器生成多项式系数表如下：

表 3-2 FEC 编码器生成多项式系数表  $g_i$ (十进制)

$i$	0	1	2	3	4	5	6	7	8
$g_i$	24	200	173	239	54	81	11	255	1

UB 支持的 FEC 工作在有限域  $GF(2^8)$ ，根据纠错能力的不同分为 RS(128,120,T=4) 和 RS(128,120,T=2) 两种模式，分别具有纠正 4 个符号和 2 个符号的能力。

编码时 RS(128,120, T=2) 模式下的 8 个校验符号与 RS(128,120, T=4) 模式下的 8 个校验符号的生成方法和数值相同。

如果使能了 FEC，任何时候发送端和对应的接收端均应使用相同的 FEC 模式。

PCS 默认支持 RS ( 128,120,T=4 ) FEC 模式，支持在链路训练阶段进行模式协商及正常工作时进行动态模式切换。

对应的 RS FEC 码字如下表所示：

表 3-3 RS FEC 码字列表

FEC 模式	N	K	T	m
RS ( 128,120,T=2 )	128	120	2	8
RS ( 128,120,T=4 )	128	120	4	8

### 3.2.2.2 Pre-FEC 分发

Pre-FEC 分发规则如下所示，其中  $m_A$ 、 $m_B$  表示不同的 FEC 编码器的消息符号，每个 FEC 编码器包含 K 个消息符号， $i = (0 : K-1)$  表示 RS FEC 消息符号的索引，每个 FEC 符号包含 m 比特。对于 RS ( 128,120 )，K=120，m=8。

PCS 支持一个编码器 ( FEC 非交织模式 ) 或者两个编码器 ( FEC 交织模式 )。

从数据链路层来的数据视为一个串行数据流( tx\_data )，当 PCS 支持两个 FEC 编码器时数据流以 RS FEC 符号粒度交织分发到两个编码器。如下公式中 CodecNum 表示编码器数量。

- 当 **CodecNum=1** 时 ( FEC 非交织模式 ):

$$m_A < K-1-i > = tx\_data <(m*i+m-1) : (m*i)>$$

- 当 **CodecNum=2** 时 ( FEC 交织模式 ):

$$m_A < K-1-i > = tx\_data <(2m*i+m-1) : (2m*i)>$$

$$m_B < K-1-i > = tx\_data <(2m*i+2m-1) : (2m*i+m)>$$

### 3.2.2.3 Post-FEC 分发

$C_A$ 、 $C_B$  是不同的 FEC 编码器输出的码字，每个 FEC 码字中包含 N 个 FEC 符号 ( 对 RS(128,120) ,N=128 )，Lane $<j,i>$ 代表编码器分发到 Lane $j$  上的第  $i$  个符号，LaneNum 表示 TX 包括的 Lane 数量。

FEC 符号分发规则如下：

- 当 **CodecNum=1** 时 ( 非交织 ):

for  $i=0$  to  $(N/LaneNum - 1)$

    for  $j=0$  to  $(LaneNum - 1)$

$$Lane < j,i > = C_A < (N-1)-i*LaneNum-j >$$

- 当 **CodecNum=2** 时 ( 交织 ):

    – if LaneNum=1:

        for  $i=0$  to  $(N*2-1)$

```

if  $i \bmod 2 = 0$ 
    Lane $<0,i>$  =  $C_A <(N-1)-i/2>$ 
if  $i \bmod 2 = 1$ 
    Lane $<0,i>$  =  $C_B <(N-1)-(i-1)/2>$ 
- if LaneNum=2/4/8:
    for  $i=0$  to  $(N*2/\text{LaneNum}-1)$ 
        for  $j=0$  to  $(\text{LaneNum}/2-1)$ 
            if  $i \bmod 2 = 0$ 
                Lane $<*>^2,i>$  =  $C_A <(N-1)-i*(\text{LaneNum}/2)-j>$ 
                Lane $<*>^2+1,i>$  =  $C_B <(N-1)-i*(\text{LaneNum}/2)-j>$ 
            if  $i \bmod 2 = 1$ 
                Lane $<*>^2,i>$  =  $C_B <(N-1)-i*(\text{LaneNum}/2)-j>$ 
                Lane $<*>^2+1,i>$  =  $C_A <(N-1)-i*(\text{LaneNum}/2)-j>$ 

```

RS ( 128,120 ) 的 Post-FEC 符号分发示意图如图 3-5 和图 3-6 所示。 $C_A$ 、 $C_B$  是不同的 FEC 编码器输出的码字，每个 FEC 码字中包含 128 个符号，由 120 个消息符号  $m_{119} \sim m_0$  和 8 个校验符号  $p_7 \sim p_0$  组成。

下图以 x8 链路示例 FEC 非交织时的符号分发：

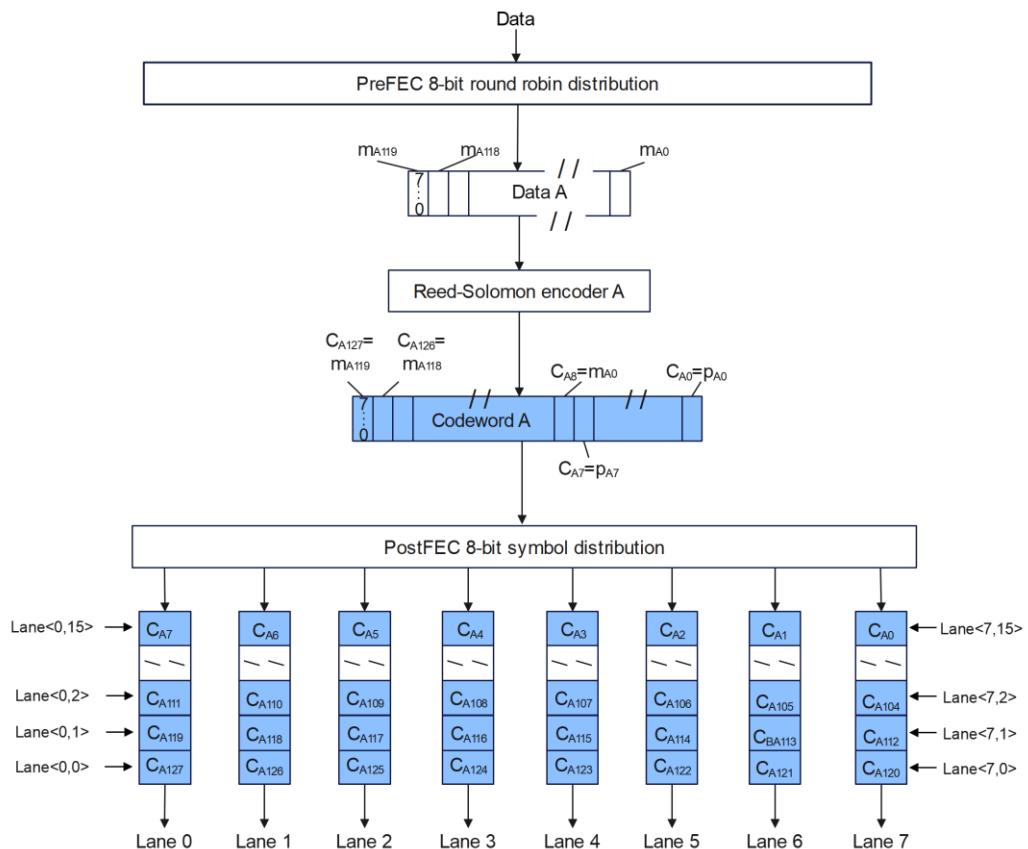


图 3-5 非交织 FEC 模式下 Post-FEC 符号到 Lane 分发示意图 ( x8 )

下图以 x8 链路示例 FEC 交织时的符号分发：

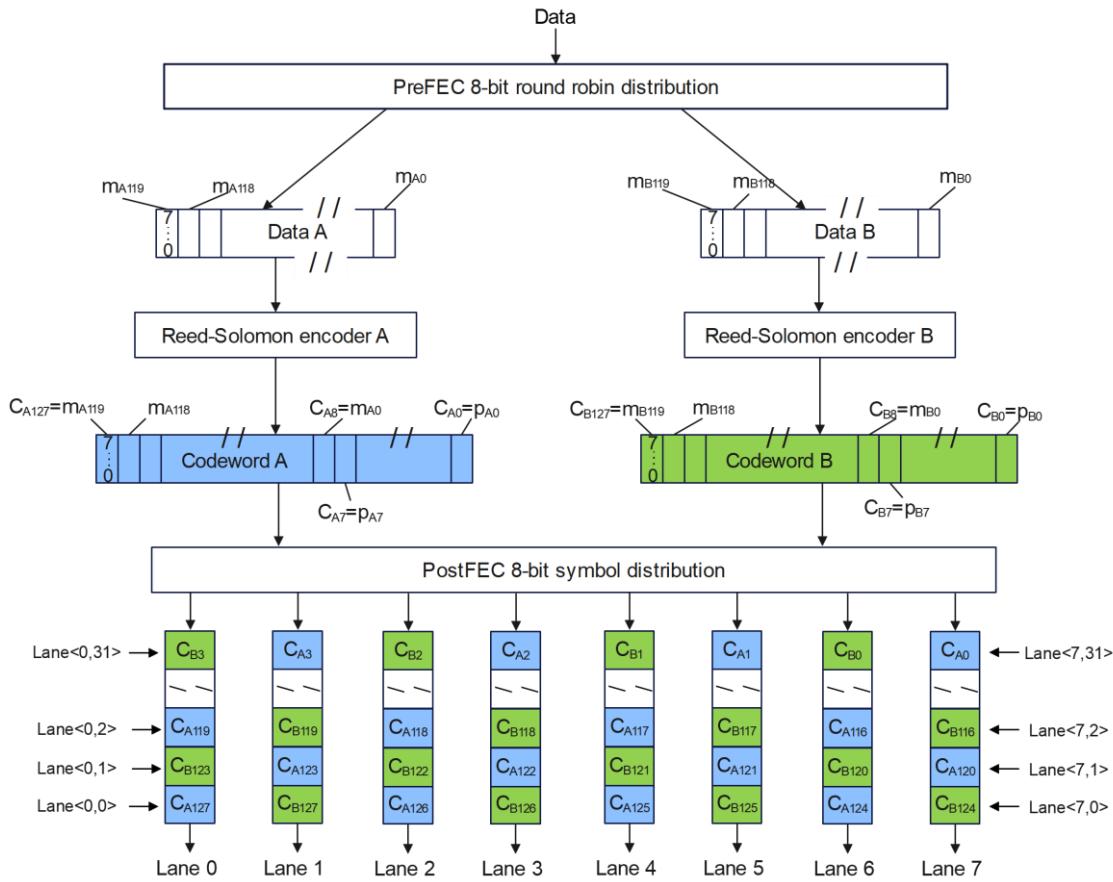


图 3-6 交织模式下 Post-FEC 符号到 Lane 分发示意图 (x8)

### 3.2.2.4 扰码

为了改善链路的电气特性，需要对数据进行加扰。

扰码方式使用加性逐 Lane 扰码，加性扰码应在发送侧及接收侧同步扰码的种子。

扰码种子值通过 AMCTL 中携带的不同的 Lane ID 值获取。

TX 和 RX 方向的扰码规则应保持一致，扰码规则如下：

- EEIB ( Exit Electrical Idle Block, 见 3.4.1.2 节 ) 的全部 symbol 不扰码。
- AMCTL 的全部 symbol 不扰码。
- LTB ( Link Training Block, 见 3.4.1.1 节 ) 的全部 symbol 都应进行扰码。
- 全部来自数据链路层的数据都应进行扰码。所有需要加扰及解扰的 symbol, lsb ( least significant bit ) 首先进行加扰或解扰，msb ( most significant bit ) 最后进行加扰或解扰。
- 非 Send\_NullBlock ( 见 3.4.3.6 节 ) 和非 Link\_Active ( 见 3.4.3.7 节 ) 状态下，收到 AMCTL with EDF ( End of Data Flit ) 之后，后续每个 AMCTL 都复位扰码种子。在 Send\_NullBlock 状态收到 AMCTL with SDF ( Start of Data Flit ) 之后的后续每个 AMCTL，以及 Link\_Active 状态的每个 AMCTL 不复位扰码种子。

### 3.2.3 接收侧编码子层功能

#### 3.2.3.1 帧锁定及 Lane 对齐

接收侧逐 Lane 接收比特流。每条 Lane 按照接收比特的顺序生成对应的数据流。PCS 首先进行 AMCTL 锁定（见 3.2.4.4 节）；由于 AMCTL 插入是周期性的，因此接收侧按 AMCTL 插入的周期进行识别和锁定。

当 AMCTL 锁定之后，接收侧根据各条 Lane 收到 AMCTL with SDF 的时序对各条 Lane 进行对齐处理，消除 Lane 之间的 skew。PCS 在不同速率下支持的最大 skew 如下表所示。

表 3-4 不同链路速率下 RX Lane 之间最大 skew 要求

<b>Data rate ( Gbps )</b>	4.0	2.578125	25.78125	53.125	106.25
<b>Maximum Skew ( ns )</b>	20	10	10	8	6

注：自定义速率的最大 skew 要求由厂家自定义。

#### 3.2.3.2 解扰

接收侧根据扰码规则对数据进行解扰。除 AMCTL 和 EEIB 外，其他 symbol 都需要输入到解扰电路中解扰，恢复原始数据。

#### 3.2.3.3 全乱序

PCS Lane 的接收序号与发送侧的发送序号可以不同。AMCTL 中携带数据码流的 Lane 序号识别信息 AMCTL.LID，接收侧 PCS 根据 AMCTL.LID 对接收数据序列进行重新排序，然后将接收码流映射到相应的 PCS Lane 上去。

只有当发生全乱序时才在接收侧使用 AMCTL.LID 进行 Lane 重排。

全乱序定义为：Lane0-LaneN 不按{0,1,2,⋯,N}或者{N,⋯,2,1,0}这样的顺序排列，或者 Lane0 不在物理通道{0,1,3,7}上。

#### 3.2.3.4 解交织

当所有码流完成 Deskev 以及重排序之后，在 FEC 被使能的情况下 PCS 根据 FEC 译码器数量将接收到的数据分发到不同的 FEC 译码器中进行 FEC 译码。

#### 3.2.3.5 FEC 解码

RS FEC 译码器根据 FEC 码字长度接收一个完整的 FEC 帧（对于 RS ( 128, 120 )，一个完整的 FEC 帧应包含 128 个 Symbol），进行纠错译码，并删除发送侧生成的校验位。

当 RS(128,120)工作在纠错能力 T=4 的模式下, 也写作 RS(128,120,T=4), 接收端接收到的 128 个符号对应的多项式为  $r(x)=r_{127} \cdot x^{127} + r_{126} \cdot x^{126} + \dots + r_1 \cdot x + r_0$ , 其中  $r_i$  表示收到的第  $i$  个符号 ( $0 \leq i \leq 127$ ),  $r_{127}$  代表首个收到的符号,  $r_0$  代表最晚收到的符号。解码过程首先计算 8 个校正子, 即第  $j$  个校正子  $S_j$  利用公式  $S_j = r(\alpha^j)$  计算 ( $0 \leq j < 8$ )。

当 RS(128,120)工作在纠错能力 T=2 的模式下, 也写作 RS(128,120,T=2) (即只具有 2 个符号的纠错能力), 在接收端计算校正子时仅计算 4 个校正子即  $S_j = r(\alpha^j)$ , 其中 ( $0 \leq j < 4$ ); 另外 4 个校正子  $S_j = r(\alpha^j)$  在解码完成之后用于验证错误是否被正确纠正时使用, 其中 ( $4 \leq j < 8$ )。

当码字所含错误符号数不大于  $T$  时, RS FEC 译码器应具备对全部错误符号的纠错能力。当码字所含错误符号数大于  $T$  时, RS FEC 译码器应给出错误符号数超过纠错能力的指示, 用于标识 FEC 纠错失败。FEC 纠错失败会触发数据链路层重传机制, 重传失败将触发物理层链路管理状态机进入 Retrain 状态。

FEC 译码器可提供链路质量监测功能。当启用此功能时, 使能错误符号计数器对 FEC 译码失败次数进行统计。当任一 FEC 译码器检测到错误符号数量超过其纠错能力时, 更新错误符号计数器 (每次 FEC 译码失败时, 计数器累加  $T+1$ ,  $T$  为 FEC 码字纠错能力), 计数器应统计全部 FEC 译码器的错误符号数。可以设置 `hi_FEC_BER` 变量用于指示链路质量, 当统计的错误符号超过规定阈值时 (阈值由具体实现决定), `hi_FEC_BER` 置 1, 表示 FEC 纠错能力不足, 系统可根据该信号进行 FEC 模式的切换。反之, 若长期无错误符号 (时间间隔由具体实现决定), 表明链路质量非常好, 或者 FEC 纠错能力充足或过剩, 系统可尝试切换为更轻的 FEC 模式或者 FEC Bypass。

### 3.2.3.6 Post-FEC 交织

FEC 译码器的输出数据按照  $m$ -bits 为粒度交织合并为一个数据流, 恢复发送侧的原始码流, 发送到数据链路层。其中  $m$  为每个 RS FEC 符号中所含的比特数 (对于 RS (128,120),  $m=8$ )。

## 3.2.4 帧定界与控制码字

UB 物理层的帧定界与控制码字 AMCTL ( Alignment Marker Control ) 用于完成帧定界, 以及扩展多种控制功能。

其主要功能如下:

- 每条 Lane 的帧定界
- Lane ID 信息识别
- 动态链路宽度切换控制
- AMCTL 插入周期控制
- 开始发送 Flit 标识 (AMCTL with SDF)
- 结束发送 Flit 标识 (AMCTL with EDF)
- FEC 控制指示
- 进入电气空闲状态指示

PCS 应在发送方向周期性插入 AMCTL。

### 3.2.4.1 eBCH-16 编码

由于不在 FEC 覆盖范围内，AMCTL 应采用具有检错和纠错能力的编码。

AMCTL 使用 eBCH-16 编码，eBCH 全称为 extended Bose-Chaudhuri-Hocquenghem，简写为 eBCH 编码。

eBCH-16 编码具有检错和纠错能力，每个 eBCH-16 码字具有如下能力：

- 检错能力，可检测 7 比特错误。
- 纠错能力，可纠正任意 3 比特随机错误及部分 4 比特随机错误。

每个 eBCH-16 码字由 BCH(15,5) 和 1 比特偶校验构成，其中 BCH(15,5) 的净荷部分为 5 比特，码长 15 比特，eBCH 编码的生成多项式表示为  $g_{BCH(15,5)}(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ 。

每个 eBCH-16 码字包含 16 比特，分为 Byte\_A 和 Byte\_B，其格式如下图所示。

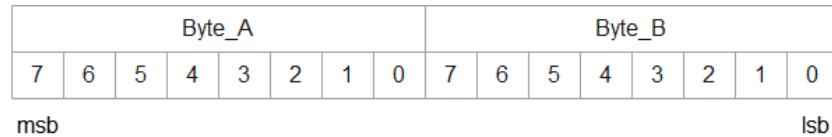


图 3-7 eBCH-16 码字格式

整个 eBCH 码字集合共 32 个，任意两个码字之间的汉明距离为 8，如下表所示。

表 3-5 eBCH(16,5)码字集合

	Byte_A								Byte_B							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CW0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CW1	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1
CW2	0	0	0	1	0	1	0	0	1	1	0	1	1	1	0	1
CW3	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
CW4	0	0	1	0	0	0	1	1	1	0	1	0	1	1	0	0
CW5	0	0	1	0	1	0	0	1	1	0	1	1	1	0	0	1
CW6	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1
CW7	0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	0
CW8	0	1	0	0	0	1	1	1	0	1	0	1	1	0	0	0
CW9	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1	1
CW10	0	1	0	1	0	0	1	1	0	1	1	1	0	0	0	1
CW11	0	1	0	1	1	0	0	1	0	0	0	1	1	1	1	0

	Byte_A								Byte_B							
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CW12	0	1	1	0	0	1	0	0	0	1	1	1	1	0	1	0
CW13	0	1	1	0	1	1	1	0	0	0	0	1	0	1	0	1
CW14	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1	1
CW15	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0
CW16	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1
CW17	1	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0
CW18	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1	0
CW19	1	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
CW20	1	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
CW21	1	0	1	0	1	1	0	0	1	0	0	0	1	1	1	0
CW22	1	0	1	1	0	0	1	0	0	0	1	1	1	1	0	0
CW23	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1
CW24	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1
CW25	1	1	0	0	1	0	0	0	1	1	1	1	0	1	0	0
CW26	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1	0
CW27	1	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
CW28	1	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
CW29	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1	0
CW30	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0
CW31	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

### eBCH-16 码字选取及位序调整

在 NRZ 和 PAM4 调制下，用于 AMCTL 的 eBCH-16 码字均应满足 symbol 直流均衡原则，即每个 symbol 的 0 或者 1 的个数（NRZ）或者负电平和正电平（PAM4）的数量相等。PAM4 调制下应使用格雷编码。

从“eBCH ( 16,5 ) 码字集合”表列出的全部 eBCH ( 16,5 ) 码字集合可选取 8 个满足直流均衡的比特序列，每个码字的高 8 比特和低 8 比特包含的 0,1 数目相等。eBCH ( 16,5 ) 码字在 NRZ 调制下天然满足直流均衡。

eBCH ( 16,5 ) 码字对原始比特顺序进行调整，使其在 NRZ 和 PAM4 下都满足直流均衡，调整规则如下所示：

Byte\_A\_new[7] = Byte\_A\_orig[4]

Byte\_A\_new[6] = Byte\_A\_orig[3]

```

Byte_A_new[5] = Byte_A_orig[2]
Byte_A_new[4] = Byte_A_orig[7]
Byte_A_new[3] = Byte_A_orig[5]
Byte_A_new[2] = Byte_A_orig[1]
Byte_A_new[1] = Byte_A_orig[6]
Byte_A_new[0] = Byte_A_orig[0]
Byte_B_new[7] = Byte_B_orig[7]
Byte_B_new[6] = Byte_B_orig[5]
Byte_B_new[5] = Byte_B_orig[4]
Byte_B_new[4] = Byte_B_orig[3]
Byte_B_new[3] = Byte_B_orig[2]
Byte_B_new[2] = Byte_B_orig[1]
Byte_B_new[1] = Byte_B_orig[6]
Byte_B_new[0] = Byte_B_orig[0]

```

用于 AMCTL 的 8 个 eBCH-16 码字及其位序调整如下表所示：

表 3-6 eBCH-16 码字位序调整表

No.	eBCH Code	Original BCH Code	NRZ	PAM4
		(76543210_76543210)	Orig: (43275160_75432160) New: (76543210_76543210)	Symbol # (3210_3210)
0	CW8	01000111_10101100	00100111_11011000	0312_2130
1	CW28	11100001_01001101	00011011_00011011	0132_0132
2	CW3	00011110_10110010	11100100_11100100	2310_2310
3	CW23	10111000_01010011	11011000_00100111	2130_0312
4	CW9	01001101_11000011	01100011_10000111	1302_3012
5	CW21	10101100_10001110	01111000_10011100	1230_3120
6	CW10	01010011_01110001	10000111_01100011	3012_1302
7	CW22	10110010_00111100	10011100_01111000	3120_1230

### eBCH-16 纠错能力

eBCH(16,5)在 BCH(15,5)基础上增加了 1 比特偶校验位，具备 3 比特纠错能力，特定情况下可纠正 4 比特错误。下图给出了全部可纠正的错误分布示意，说明如下：

1. 整个 eBCH 码字共 16 比特，任意位置只出现 1 比特错误，可纠。
2. 整个 eBCH 码字共 16 比特，任意位置只出现 2 比特错误，可纠。
3. 整个 eBCH 码字共 16 比特，任意位置只出现 3 比特错误，可纠。

4. 整个 eBCH 码字共 16 比特，其中 BCH 码字部分（即 eBCH 码字前 15 比特）只出现 3 比特错误，同时 1 比特偶校验位置也发生错误，可纠。

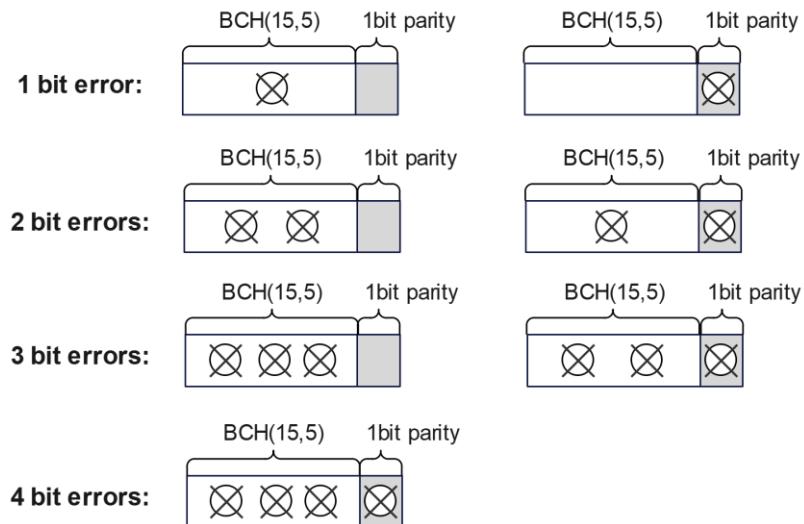


图 3-8 eBCH-16 全部可纠正的错误图示

### 3.2.4.2 AMCTL 结构

一个 AMCTL 长度为 40 个 symbol，由 BODY、END、LID、CTRL\_TYPE、CTRL\_DETAIL 组成。

- BODY 是 AMCTL 的开始域段，有 12 个 symbol，由 3 组相同的 eBCH-16 码字（CW21，CW28）组成。BODY 用于 AMCTL 锁定过程中码流的辅助识别。
- END 长度为 4 个 symbol，由 2 个 eBCH-16 码字 CW22 组成。END 在 AMCTL 锁定过程中用于确定 AMCTL。
- LID 长度为 8 个 symbol，由 2 组相同的 4 个 symbol 组成，每组 4 个 symbol 由 2 个 eBCH-16 码字组成。LID 用于识别当前的 Lane ID。
- CTRL\_TYPE 长度为 8 个 symbol，由 2 组相同的 4 个 symbol 组成，每组 4 个 symbol 由 2 个 eBCH-16 码字组成。CTRL\_TYPE 用于携带控制命令，不同控制命令种类的编码不同。在无控制命令下发时，CTRL\_TYPE 置为 No Command，接收侧识别到 No Command 后忽略 CTRL\_DETAIL 的内容。
- CTRL\_DETAIL 长度为 8 个 symbol，由 2 组相同的 4 个 symbol 组成，每组 4 个 symbol 由 2 个 eBCH-16 码字组成。CTRL\_DETAIL 用于区分每种 CTRL\_TYPE 命令的进一步分类。

AMCTL 的结构如下表所示：

表 3-7 AMCTL 域段结构表

Symbol Number	Description
0 ~ 11	BODY: CW21, CW28
12 ~ 15	END: CW22, CW22

Symbol Number	Description
16 ~ 23	LID: Lane ID Indicator
24 ~ 31	CTRL_TYPE: Control Command Type
32 ~ 39	CTRL_DETAIL: Control Command Detail

### AMCTL.LID

指示当前的 Lane ID 编号，用于获取扰码种子，全乱序功能等。

AMCTL.LID 编码如下：

表 3-8 AMCTL.LID 编码表

Lane ID Indicator	LID [63:48]	LID [47:32]	LID [31:16]	LID [15:0]
Lane0	CW3	CW3	CW3	CW3
Lane1	CW3	CW8	CW3	CW8
Lane2	CW3	CW9	CW3	CW9
Lane3	CW3	CW10	CW3	CW10
Lane4	CW3	CW21	CW3	CW21
Lane5	CW3	CW22	CW3	CW22
Lane6	CW3	CW23	CW3	CW23
Lane7	CW3	CW28	CW3	CW28
NULL	CW21	CW3	CW21	CW3

### AMCTL 控制功能

AMCTL.CTRL\_TYPE 定义了各种控制功能,AMCTL.CTRL\_DETAIL 定义了每一种控制功能的细分功能。

AMCTL 控制功能种类如下：

- **FEC Control**

用于指示此 AMCTL 后续码流的 FEC 打开或者关闭。

- **Link Width Switch Indicator**

- **X0:** 在 Link\_Active 状态指示 EDF ( End of Data Flit )。即在发送 AMCTL with EDF 之后，Link 将从发送 Flit 切换到发送 LTB。

- **x1, x2, x4, x8:**

- 在 Link\_Active 状态，在动态 Link Width 切换时指示 TX 目标切换宽度。

- 在 Send\_NullBlock 状态，指示 SDF ( Start of Data Flit )。即在发送 AMCTL with SDF 之后，Link 将从发送 LTB 切换到发送 Flit。

- **EI Indicator**

在 TX 进入电气空闲时使用。

在进入电气空闲时(比如切速场景),用于指示当前的 Lane 在发送完此 AMCTL 后会进入电气空闲状态。

UB Port 发送 AMCTL with Enter Electrical Idle ( AMCTL with EEI ) 通知对端 Port, 它的发送器将在发送 AMCTL with EEI 后进入电气空闲状态。

- **AMCTL Insert Period Control**

本规范支持不同的 AMCTL 插入周期, 除了正常的 AMCTL 插入周期 Period0 (通过用户自定义寄存器配置)之外, 本规范定义了用于快速动态链路宽度切换 ( QDLWS ) 升 Lane 过程中用到的 AMCTL 插入短周期 Period1。当 AMCTL CTRL\_TYPE == AMCTL Insert Period Control, 并且 AMCTL CTRL\_DETAIL == Period1 时, 表示后继码流中 AMCTL 的插入周期为短周期。短周期值通过用户自定义寄存器配置。

- **Remote TX Link Width Switch Indicator**

此控制模式用于在 Link\_Active 状态下强制请求对端 TX 快速降 Lane, 降 Lane 过程不进入 Retrain 状态。详细内容和过程见 3.4.2.4 节。

AMCTL CTRL\_TYPE 编码定义如下表:

**表 3-9 AMCTL CTRL\_TYPE 编码功能映射表**

CTRL_TYPE[63:48]/ CTRL_TYPE[31:16]	CTRL_TYPE[47:32]/ CTRL_TYPE[15:0]	Function
CW3	CW8	FEC Control
CW8	CW9	Link Width Switch Indicator
CW9	CW10	EI Indicator
CW10	CW21	AMCTL Insert Period Control
CW21	CW22	Remote TX Link Width Switch Indicator
CW22	CW23	Reserved
CW23	CW28	Reserved
CW28	CW3	No Command

AMCTL CTRL\_DETAIL 编码如下表:

**表 3-10 AMCTL CTRL\_DETAIL 编码功能映射表**

CTRL_TYPE	CTRL_DETAIL[63:48] / CTRL_DETAIL [31:16]	CTRL_DETAIL[47:32] / CTRL_DETAIL [15:0]	详细分类
FEC Control	CW3	CW9	Reserved
	CW8	CW10	Reserved
	CW9	CW21	Reserved
	CW10	CW22	Reserved

CTRL_TYPE	CTRL_DETAIL[63:48] / CTRL_DETAIL [31:16]	CTRL_DETAIL[47:32] / CTRL_DETAIL [15:0]	详细分类
Link Width Switch Indicator	CW21	CW23	Reserved
	CW22	CW28	Reserved
	CW23	CW3	Dynamic Close FEC
	CW28	CW8	Dynamic Open FEC
EI Indicator	CW3	CW9	X0 ( End of Data Flit )
	CW8	CW10	x1
	CW9	CW21	x2
	CW10	CW22	x4
	CW21	CW23	x8
	CW22	CW28	Reserved
	CW23	CW3	Reserved
	CW28	CW8	Reserved
AMCTL Insert Period Control	CW3	CW9	Reserved
	CW8	CW10	Reserved
	CW9	CW21	Reserved
	CW10	CW22	Reserved
	CW21	CW23	Reserved
	CW22	CW28	Enter Electrical Idle ( EEI )
	CW23	CW3	Reserved
	CW28	CW8	Reserved
Remote TX Link Width Switch Indicator	CW3	CW9	Reserved
	CW8	CW10	x1
	CW9	CW21	x2
	CW10	CW22	x4

CTRL_TYPE	CTRL_DETAIL[63:48] / CTRL_DETAIL [31:16]	CTRL_DETAIL[47:32] / CTRL_DETAIL [15:0]	详细分类
	CW21	CW23	x8
	CW22	CW28	Reserved
	CW23	CW3	Reserved
	CW28	CW8	Reserved
No Command	CW3	CW9	No Command

### AMCTL 识别机制

由于两个相邻 AMCTL 之间的数据长度为定值，接收侧定期检测 END。AMCTL 采用 eBCH-16 编码，每个码字都有一定的纠错能力。

当接收到的 END 在经过纠错后和期望值相同时，接收侧认为接收到了正确的 AMCTL。

当接收到的 END 的错误比特数量超过 eBCH-16 编码的纠错能力时，接收侧认为未接收到正确的 AMCTL，不对其进行任何操作。如果链路处于 Link\_Active 或者 Send\_NullBlock 状态时，可能会触发链路管理状态机进入 Retrain 状态进行链路训练，滑码重新锁定 AMCTL；如果链路处于别的状态时，会滑码进行 AMCTL 的重新锁定，参考 AMCTL 锁定流程。

LID[63:0]分为 LID[63:32]和 LID[31:0]，两部分内容应完全一致；若收到的 AMCTL 中 LID[63:32]和 LID[31:0]不一致，则认为是无效 LID。在 AMCTL 锁定后会记录经过 eBCH-16 纠错后的 LID，如果连续收到两个 AMCTL，经过纠错后的 LID 一致，则刷新当前 Lane 的 Lane ID 值。LTB 中的 Lane ID 也可用于判断当前 Lane 的 ID 值，具体使用哪种方式由实现决定。

CTRL\_TYPE[63:0]分为 CTRL\_TYPE[63:32]和 CTRL\_TYPE[31:0]，两部分定义完全一样。只要 CTRL\_TYPE[63:32]和 CTRL\_TYPE[31:0]中任意一个经过 eBCH-16 纠错后成功匹配有效编码，则可认为其为有效值。如果两部分都匹配了有效编码，但是两部分内容不一致，接收端不执行控制动作。

CTRL\_DETAIL[63:0]分为 CTRL\_DETAIL[63:32]和 CTRL\_DETAIL[31:0]，两部分定义完全相同。只要 CTRL\_DETAIL[63:32]和 CTRL\_DETAIL[31:0]中任意一个经过 eBCH-16 纠错后成功匹配有效编码，则可认为其为有效值。若两部分都匹配了有效编码，但内容不一致，接收端不执行控制动作。

### 3.2.4.3 发送侧 AMCTL 处理规则

- 发送侧同一个 Link 中所有 Lane 上的数据传输应工作在同一个时钟频率下。
- 发送侧周期性地在每条 Lane 上同时插入长度为 40 个 symbol 的 AMCTL。
- 链路管理状态机处于 Send\_NullBlock 和 Link\_Active 状态时的 AMCTL 插入规则：
  - 进入 Send\_NullBlock 状态时首先发送一个 AMCTL with SDF，之后周期性插入 AMCTL；

- 在 Link\_Active 状态下进行动态升 Lane 时，在发送 RLTB 前需将 AMCTL 插入周期协商切换为 Period1，其周期值可通过用户自定义寄存器配置，默认值是 640 个 symbol 插入一个 AMCTL（配置值需保证一个连续的 LTB 不被打断）。在升 Lane 期间，发送数据的 Lane 和发送 LTB 的 Lane 需按该周期定期插入 AMCTL。在完成升 Lane 后，必须将 AMCTL 插入周期协商切换为升 Lane 之前的周期。
- 在其它链路训练状态，每隔 32 个 LTB 发送 1 个 AMCTL。
- AMCTL 在 LTB 边界插入。
- AMCTL 不参与 FEC 编码。
- AMCTL 不参与扰码和预编码。

#### 3.2.4.4 接收侧 AMCTL 处理规则

- 接收侧根据发送侧 AMCTL 插入规则对 AMCTL 进行检测及锁定，如图 3-9 示。
- 接收侧通过识别 END 来判断 AMCTL 的边界。
- 接收侧通过设置计数器对相邻两个 AMCTL 之间比特流进行计数。若 AMCTL 插入间隔是 640 个 symbol，每个 symbol 有 8 个比特，相邻两个 AMCTL 之间的数据长度是 5120 比特，计数器的计数范围为[0: 5120]，计数器从 0 开始，当计数累加到 5120 时停止计数。当接收侧识别到 END 时，计数器清零，并从 END 后 24 个 symbol 处开始重新计数。

##### 实现参考：AMCTL 锁定流程

在链路训练期间及 Link\_Active 状态下发送器在每条 Lane 上周期性发送 AMCTL，因此正常情况下接收端 AMCTL 应该始终处于锁定状态。

每条 Lane 均应进行 AMCTL 锁定，下图是 AMCTL 锁定流程示意：

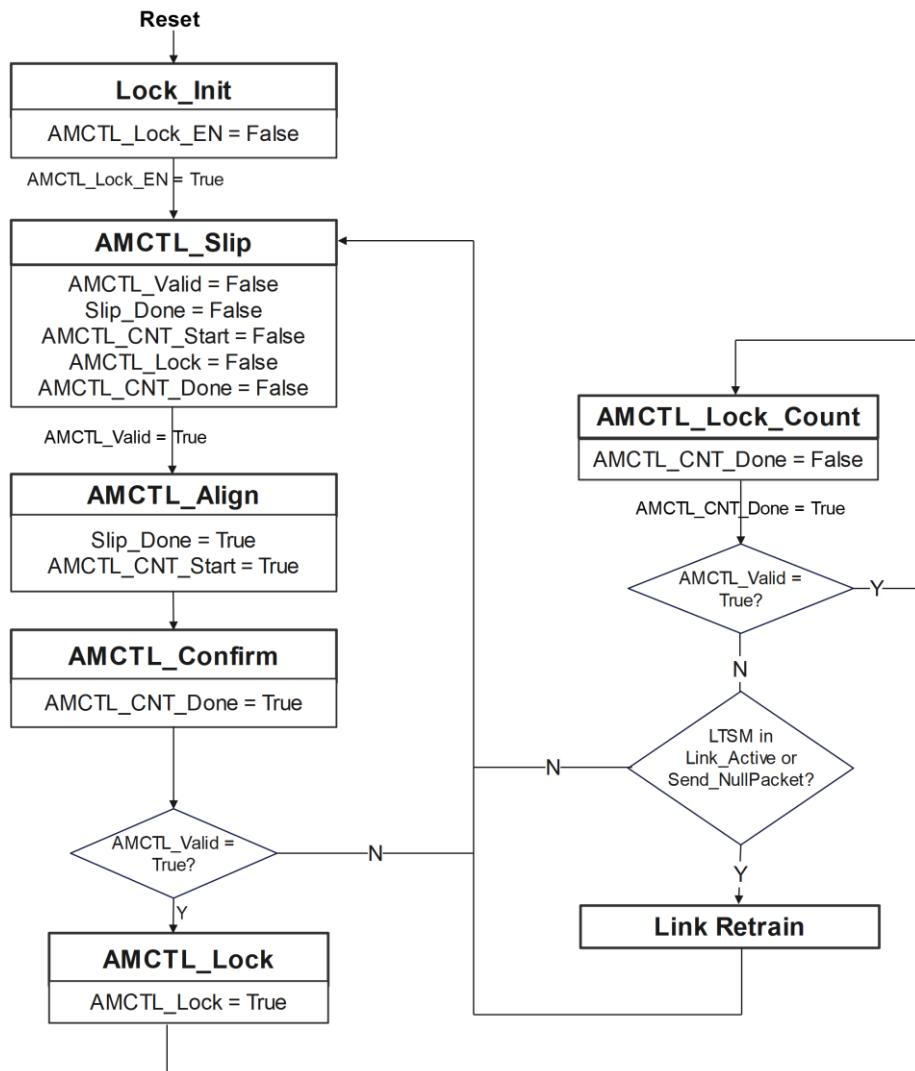


图 3-9 AMCTL 锁定参考流程图

**AMCTL 锁定过程涉及的状态：**• **Lock\_Init**

开始 AMCTL 检测前的初始状态，在此状态中所有变量默认为 `False`。当链路训练开始时，比如进入 Discovery 状态时，变量 `AMCTL_Lock_EN` = `True`，进入下一个状态 `AMCTL_Slip`。

• **AMCTL\_Slip**

在此状态中，除 `AMCTL_Lock_EN` 之外的其它变量都复位为 `False`。开始在接收的比特流中滑码寻找 END，找到一个有效的 AMCTL 后，变量 `AMCTL_Valid` = `True`，进入下一个状态 `AMCTL_Align`。

此状态中 AMCTL 有效的判断是：END 正确，END 之前的那个 BODY 可以作为辅助判断依据。

• **AMCTL\_Align**

此状态是一个临时状态，表示找到了第一个 AMCTL 识别符。滑码寻找过程完成，变量 `Slip_Done` = `True`。在 END 之后再经过 24 个 symbol，变量 `AMCTL_CNT_Start` = `True`，启动计数器计数，

以便检查在规定的间隔处 AMCTL 是否再次按预期出现。在链路训练过程中，AMCTL 间隔是 32 个 LTB ( 512 个 symbol, 4096 比特 )，计数器记到 4096 时停止，进入下一个状态 AMCTL\_Confirm。

- AMCTL\_Confirm

在此状态中，变量 AMCTL\_CNT\_Done = True。检查是否收到有效的 AMCTL。

- 如果收到了若干个 ( 数量可配置 ) 有效的 AMCTL，进入下一个状态 AMCTL\_Lock，完成 AMCTL 锁定。
- 如果没有收到有效的 AMCTL，跳转到 AMCTL\_Slip 状态。

- AMCTL\_Lock

此状态为 AMCTL 锁定状态，是临时状态，设置变量 AMCTL\_Lock = True 后跳转到 AMCTL\_Lock\_Count 状态。

- AMCTL\_Lock\_Count

在此状态在各条 Lane 上周期性地检查在期望位置是否收到有效的 AMCTL。

- 链路训练状态，间隔周期是 512 个 symbol。有效 AMCTL 的判断是 END 正确，BODY 作为辅助判断 ( 比如 BODY 错误时只记录错误，不影响 AMCTL 锁定状态 )。
- 在 Send\_NullBlock 或 Link\_Active 状态，根据间隔周期在期望的位置检查 AMCTL。若未收到有效的 END，即认为收到的 AMCTL 无效。
- AMCTL 有效时，在当前 AMCTL 结束后启动另一个周期计数，重复在期望的位置判断是否收到有效的 AMCTL。
- 如果在连续若干个 ( 数量可配置 ) 期望的位置没有收到有效的 AMCTL：
  - 若此时处于 Link\_Active 或 Send\_NullBlock 状态，跳转到链路管理状态机的 Retrain 状态。
  - 若此时处于其它链路训练状态，跳转到 AMCTL 锁定参考流程的 AMCTL\_Slip 状态。

### 3.2.5 通道符号分发与位序

#### 3.2.5.1 Lane 上的 Symbol 传输位序

##### NRZ 模式

NRZ 模式下每个 symbol 包括 8 比特 ( bit7 ~ bit0 )，其中 bit0 是 lsb，bit7 是 msb。每个 symbol ( 包括数据、LMB 和 AMCTL ) 发到 Lane 上的顺序是 bit0, bit1, … bit7。

NRZ 模式下的 symbol 在 Lane 上的传输位序如下图所示：

### 3 物理层

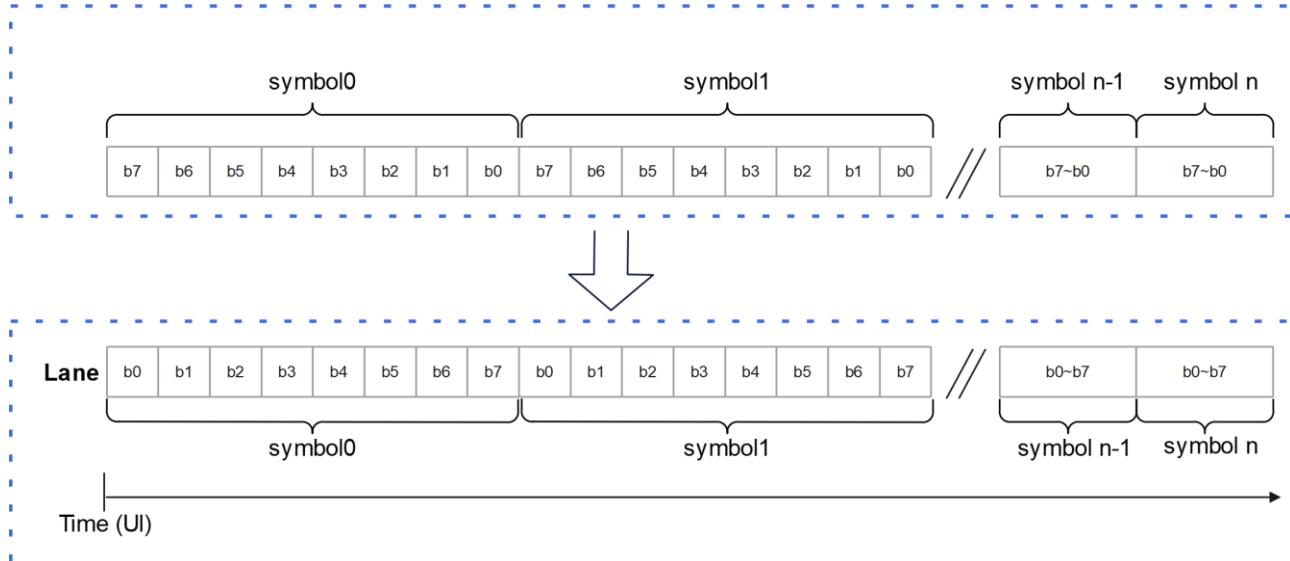


图 3-10 NRZ 模式下 Symbol 在 Lane 上的传输位序图

注: NRZ 模式下每个 UI 包括 1 个比特。

### PAM4 模式

PAM4 模式下每个 symbol (包括数据, LMB 和 AMCTL) 包括 8 比特 (bit7 ~ bit0); bit0 是 lsb。

PAM4 模式下的 symbol 在 Lane 上的传输位序如下图所示:

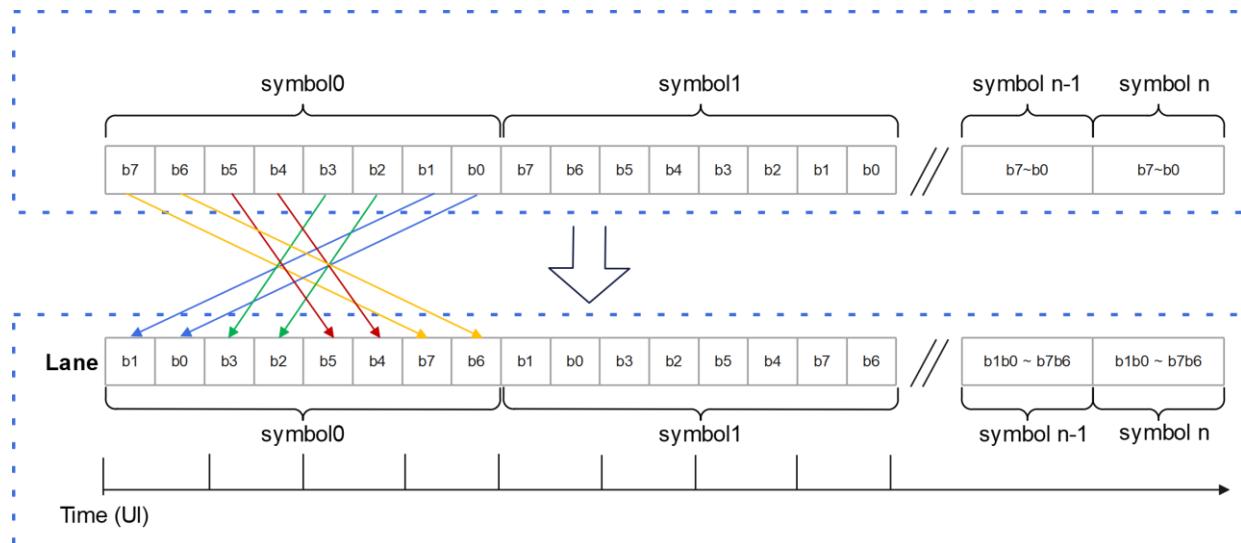


图 3-11 PAM4 模式下 Symbol 在 Lane 上的传输位序图

注: PAM4 模式下每个 UI 包括 2 个比特。

#### 3.2.5.2 AMCTL 的传输位序

发送方向上的 AMCTL 长度为 40 个 symbol, 每个 symbol 长度为 8 比特。

### 3 物理层

Symbol 传输顺序为 symbol0, symbol1, symbol2 ..... symbol39。

每个 symbol 先传输 bit0，后传输 bit1…bit7。

比如：symbol0~symbol3 是 AMCTL.BODY，其对应关系如下图所示：

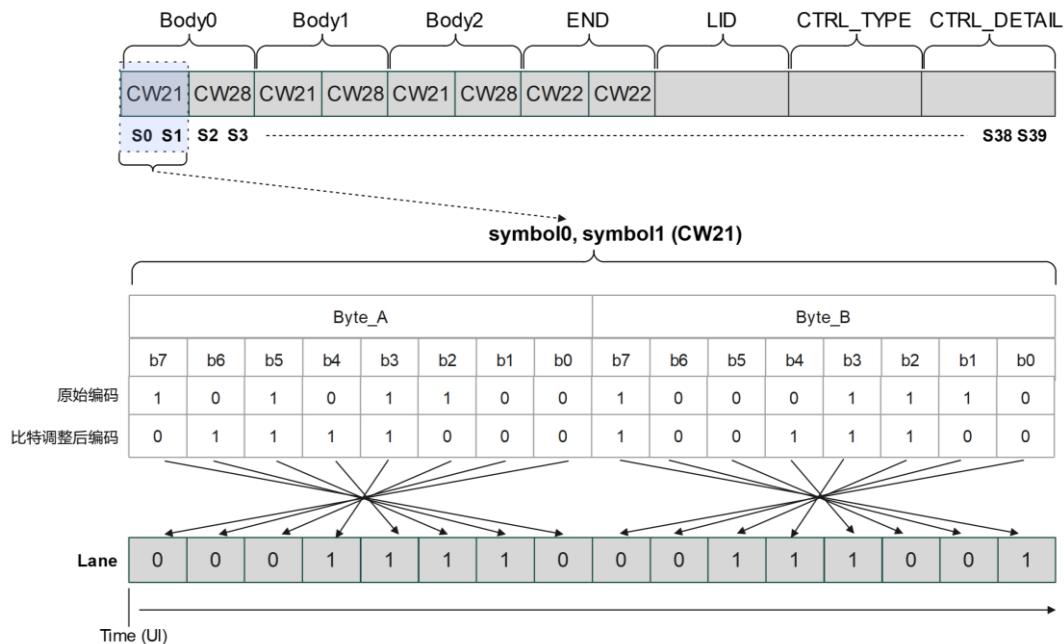


图 3-12 AMCTL NRZ 模式下传输位序图

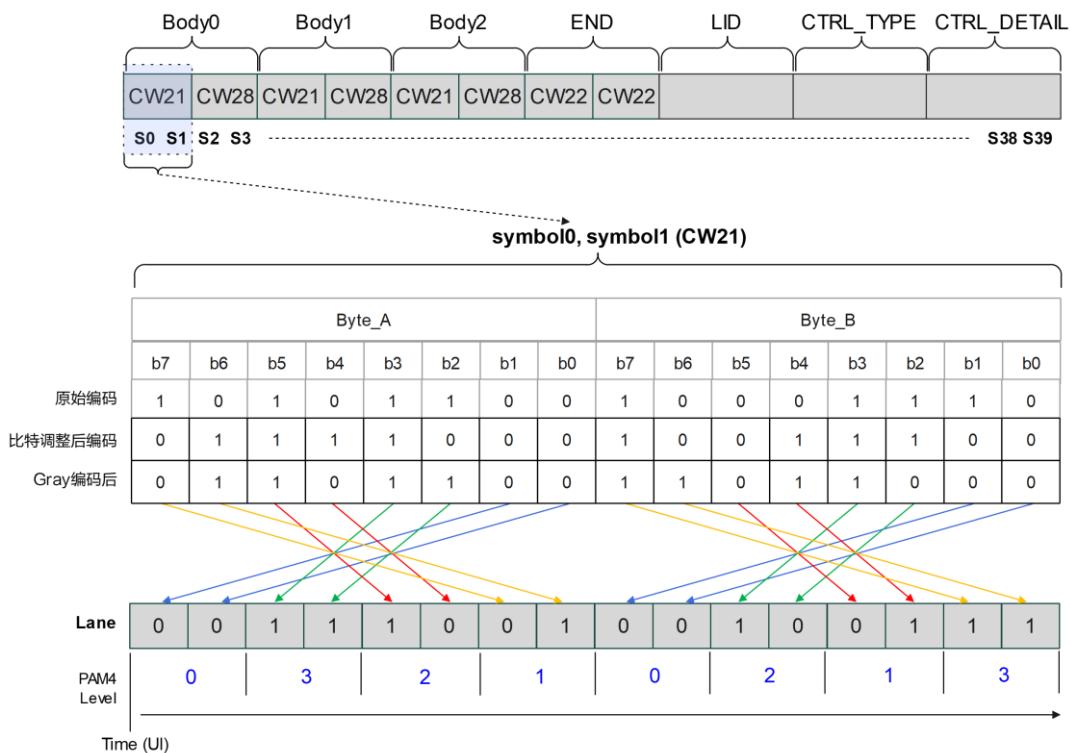


图 3-13 AMCTL PAM4 模式下传输位序图

### 3.2.5.3 LMB 的传输位序

每个 LMB (包括 LTB,EEIB。见 3.4.1 节 ) 长度为 16 个 symbol, 先传输 symbol0, 后传输 symbol1…15。

每个 symbol 长度为 8 比特, 先传输 bit0, 后传输 bit1…7。

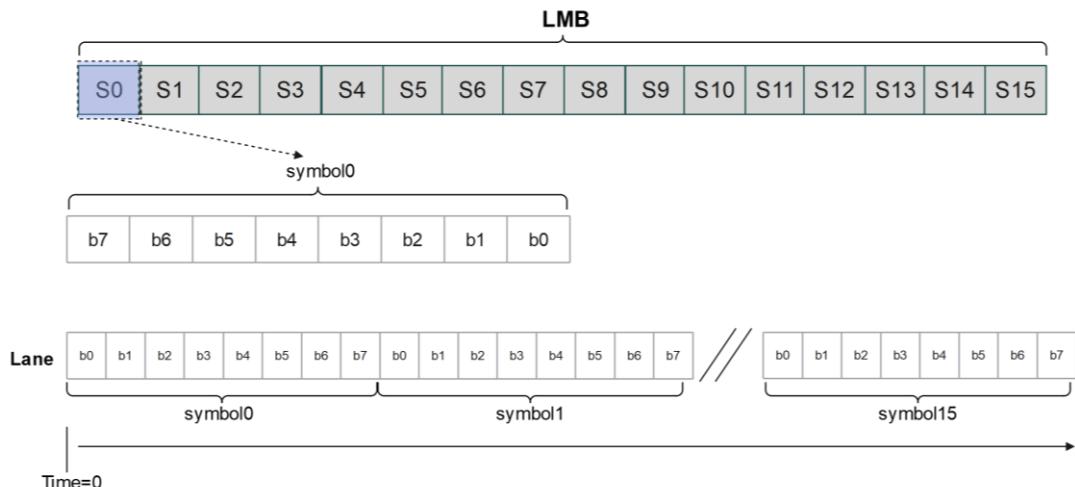


图 3-14 NRZ 模式下 LMB 传输位序图

PAM4 模式下 LMB 在 Lane 上的传输位序见图 3-11。

### 3.2.5.4 FEC Bypass 模式下 Symbol 分发

在 Send\_NullBlock 及 Link\_Active 状态下, Link 上传输 Flit。一个数据链路层 packet 包括若干个 Flit, 每个 Flit 有 20 个 symbol。上层无业务包, 或者业务包不能充满 Link 带宽时, 在 Link 上传输 Null Block (见 4.3.3.2 节)。每条 Lane 上周期性插入 AMCTL 以进行物理层帧定界。

FEC Bypass 模式下每个 symbol 的长度是一个 Byte。首先传输的是 Null Block, 在 Send\_NullBlock 状态中在 AMCTL with SDF 之后从 Lane0 开始以 symbol 为单位按顺序轮转发送到各条 Lane 上; 后续 Packet 依次发出。

除非要进行链路训练状态跳转, 否则数据中间只能间插 AMCTL。

每个 symbol 在 Lane 上的传输位序见 3.2.5.1 节。

图 3-16 以 AMCTL 间插周期为 640 个 symbol 为例, 说明 FEC Bypass 情况下在一个 x8 Link 上的数据分发方式。下图中每个 AMCTL 包括一个完整的 AMCTL 结构。

假设有 3 个数据链路层 Packet, 每个 Packet 包括若干个 symbol, 每个 Byte 由 Packet No., Flit No., Symbol No. 来表达 (比如 P0.F1.S2 代表 Packet0 的 Flit1 中的 Symbol2)。Packet0 发在 Link 上的第一个 symbol 是 P0.F0.S0, 即 LPH (Link Packet Header) 的 Byte0 (见 4.3.2.2.2 节)。

### 3 物理层

	Byte +0								Byte +1								Byte +2								Byte +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
Byte0	Px.F0.S0								Px.F0.S1								Px.F0.S2								Px.F0.S3							
Byte4	Px.F0.S4								Px.F0.S5								Px.F0.S6								Px.F0.S7							
Byte8	Px.F0.S8								Px.F0.S9								Px.F0.S10								Px.F0.S11							
Byte12	Px.F0.S12								Px.F0.S13								Px.F0.S14								Px.F0.S15							
Byte16	Px.F0.S16								Px.F0.S17								Px.F0.S18								Px.F0.S19							

图 3-15 数据链路层包第一个 Flit Symbol 示意图

每个 Null Block 包括一个 Flit, Null Block 的 Symbol 表达为 NB.Sx。

图中包括 3 个数据链路层包 Packet0, Packet1, Packet2 以及若干个 Null Block。

1. Packet0 包括 2 个 Flits, 共 40 个 symbol 依次进入 Link 传输, 表达为: P0.F0.S0, P0.F0.S1 ... P0.F0.S19, P0.F1.S0, P0.F1.S1 ... P0.F1.S19。
2. Packet1 包括 2 个 Flits, 共 40 个 symbol 依次进入 Link 传输, 表达为: P1.F0.S0, P1.F0.S1 ... P1.F0.S19, P1.F1.S0, P1.F1.S1 ... P1.F1.S19。
3. Packet2 包括 1 个 Flit, 共 20 个 symbol 依次进入 Link 传输, 表达为: P2.F0.S0, P2.F0.S1 ... P2.F0.S19。

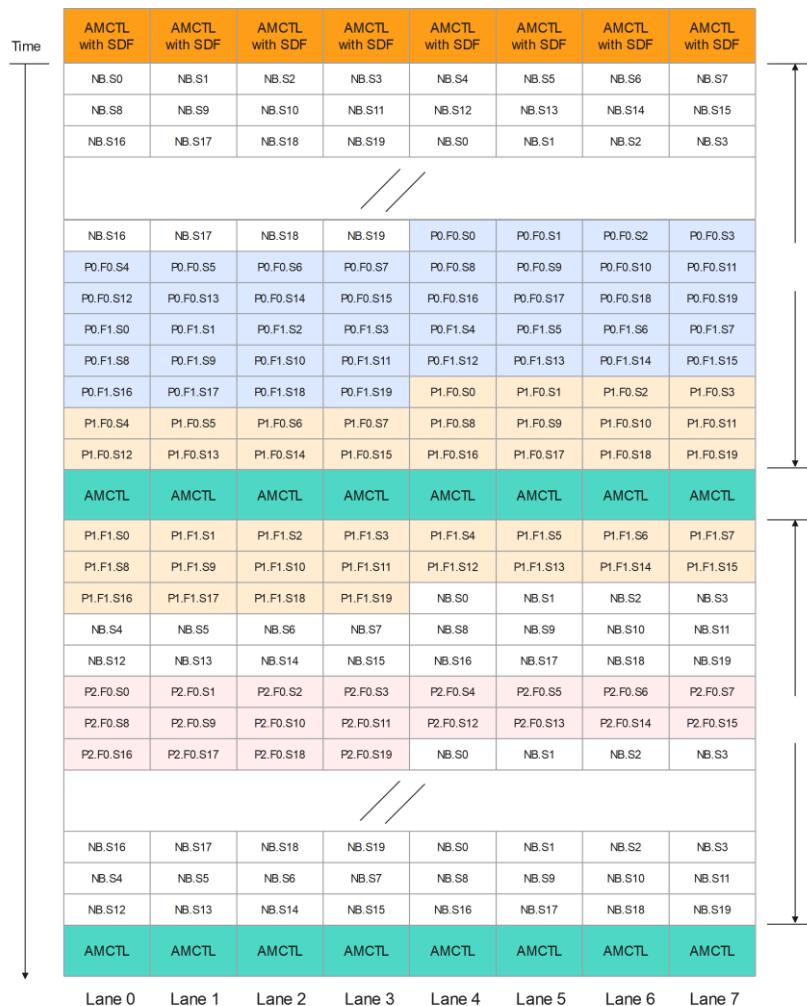


图 3-16 FEC Bypass 数据分发及成帧示意图

### 3.2.5.5 FEC 模式下 Symbol 分发

#### FEC 交织模式下 Symbol 分发

从数据链路层来的数据以及 Null Block 以 8 比特为粒度从 S0 开始顺序分发到各 FEC 编码器，FEC 编码器附加校验符号  $p_7 \sim p_0$  ( $p_7$  是 MSB) 后，各编码器将本 FEC 帧符号交织发送到各 Lane 上。

每个符号在 Lane 上的比特传输顺序见 3.2.5.1 节。

FLIT																																
	Byte0								Byte1								Byte2								Byte3							
	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
Byte0	7		S0		0				7		S1						7		S2							S3						
Byte4			S4								S5								S6							S7						
Byte8			S8								S9								S10							S11						
Byte12			S12								S13								S14							S15						
Byte16			S16								S17								S18							S19						

图 3-17 RS(128,120)FEC 模式下 Flit Symbol 示意图

以 x8 为例分发示意图如下，其中  $m_{A119} \sim m_{A0}$ 、 $m_{B119} \sim m_{B0}$  是消息符号； $p_{A7} \sim p_{A0}$ 、 $p_{B7} \sim p_{B0}$  是 FEC 的校验符号。

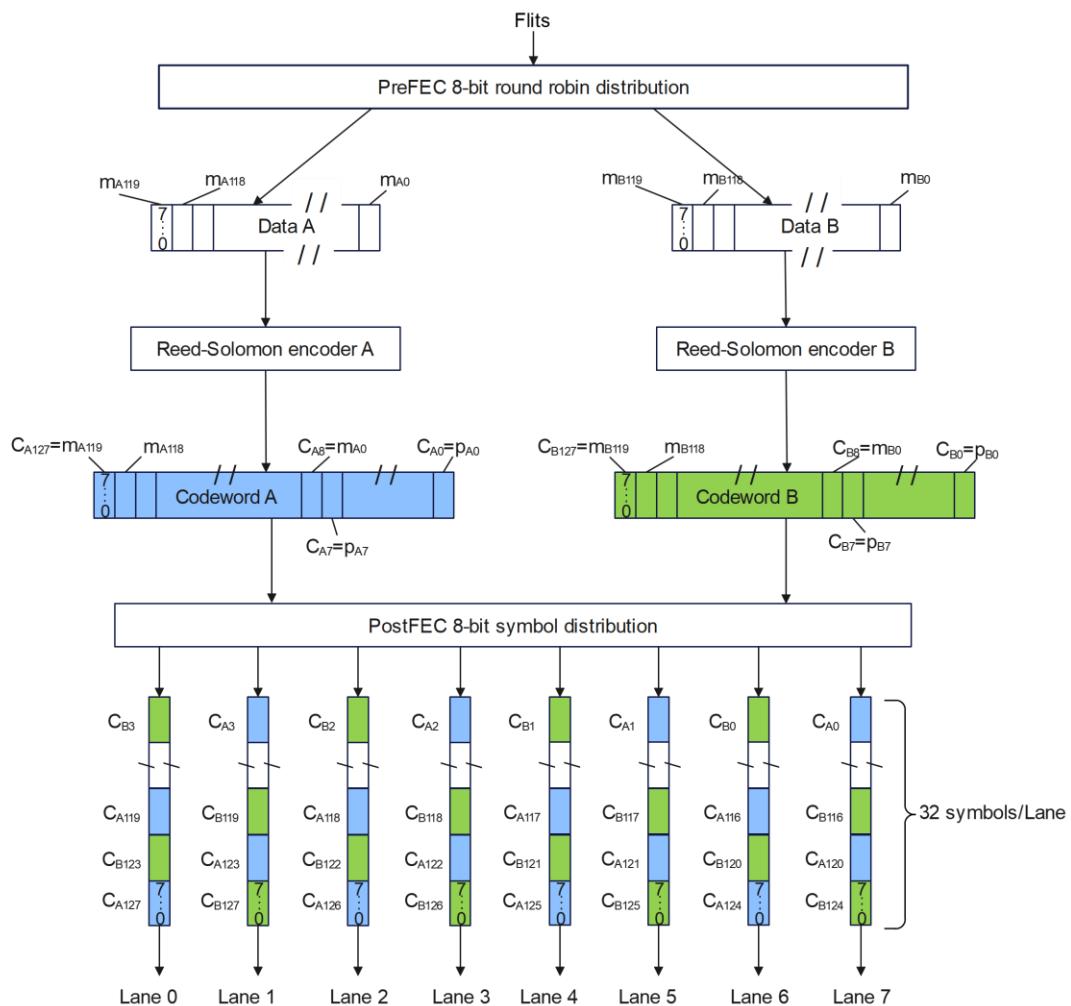


图 3-18 RS(128,120)FEC 交织在 x8 链路上符号分发示意图

### 3 物理层

上述示例中 FEC 符号在 Lane 上的成帧示意图如下：

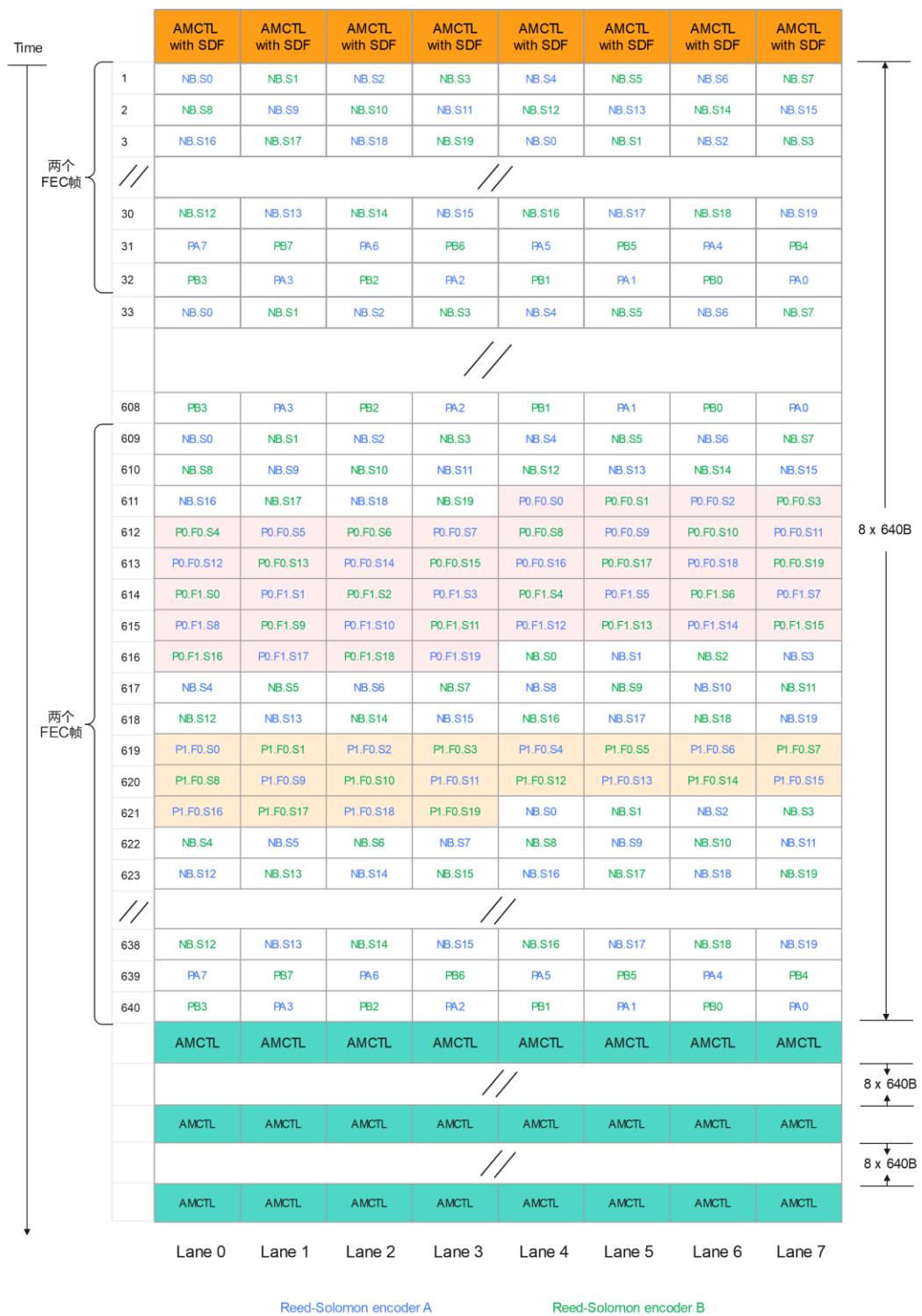


图 3-19 RS(128,120) FEC 交织模式数据分发及成帧示意图

### FEC 模式非交织模式下 Symbol 分发

FEC 非交织模式分发规则如下图所示：

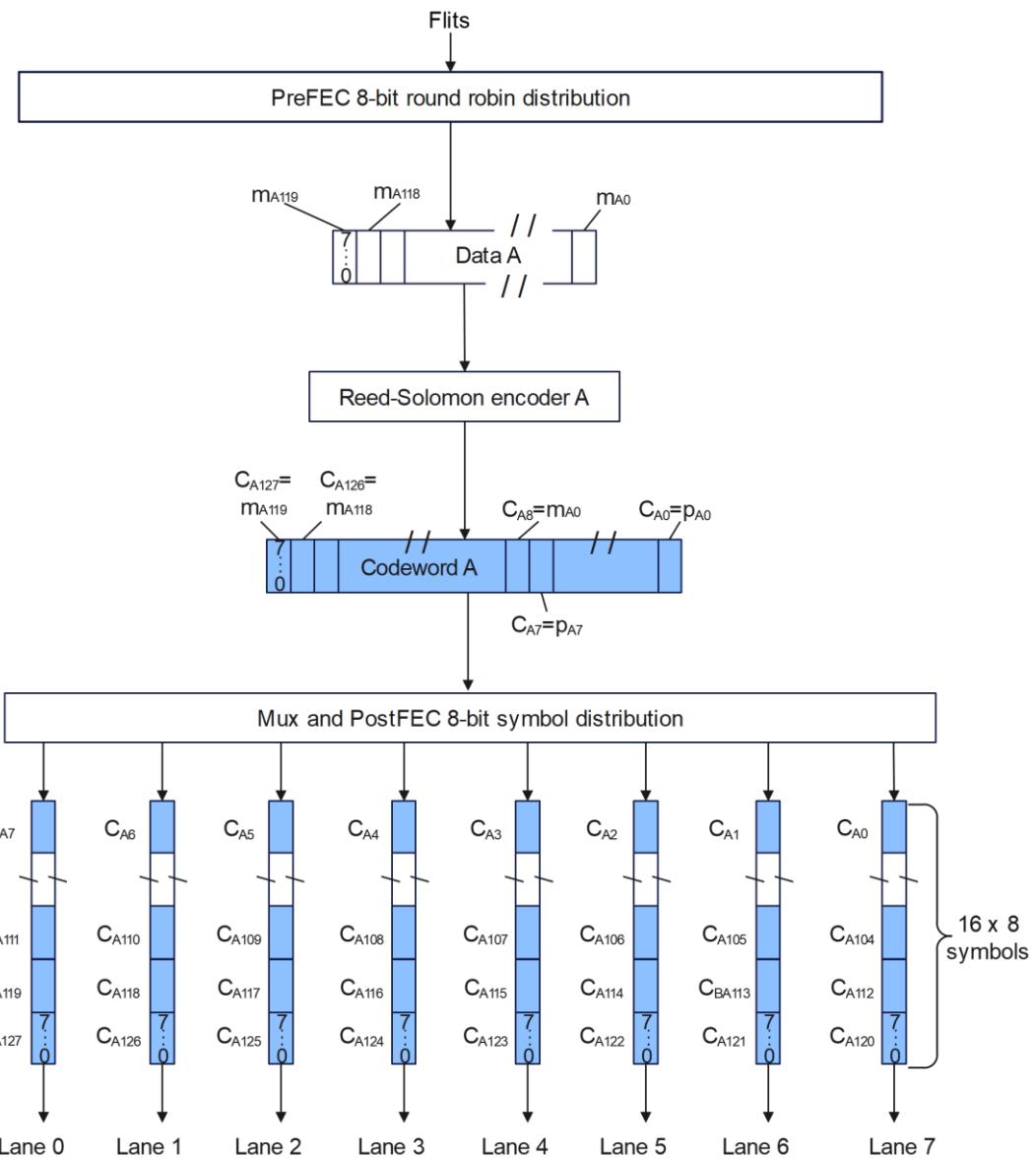


图 3-20 RS(128,120)FEC 非交织分发示意图

$m_{A119} \sim m_{A0}$  是消息符号； $p_{A7} \sim p_{A0}$  是 FEC 的校验符号。

### 3.2.6 数据通路低功耗机制

数据链路层无数据发送时，PCS 会插入 Null Block，经过 FEC 编码和加扰等处理后发送给 SerDes，用以维持对接 UBUU 的 SerDes CDR ( Clock Data Recovery ) 锁定，Null Block 的插入会增加物理层的动态功耗。

UB 提供了物理层数据通路动态功耗控制功能, PCS 可在数据链路层无数据发送时不插入 Null Block, 关闭 FEC 编码和加扰等 PCS 处理逻辑, 降低物理层动态功耗。这是一个可选特性。具体流程如下:

#### 关闭 PCS 流程:

1. PCS 检测到数据链路层无数据下发后, 排空当前数据通路的数据, 确认 PCS 内部无数据待发送。
2. 发送一个 AMCTL, AMCTL CTRL\_TYPE 域设置为 FEC Control, AMCTL CTRL\_DETAIL 置为 Dynamic Closed FEC。
3. AMCTL 发送结束后立即向 SerDes 各条 Lane 发送 PRBS23 码流, 同时关闭 PCS TX 加扰, FEC 等逻辑。发送 PRBS23 码流的目的是维持对接 UBU 的 SerDes CDR 锁定, PRBS23 的多项式和扰码保持一致。在发送 PRBS23 码流期间, 仍然周期性插入 AMCTL。
4. 对端接收侧收到 AMCTL 并对内容进行解析, 如果 AMCTL CTRL\_TYPE == FEC Control 同时 AMCTL CTRL\_DETAIL == Dynamic Closed FEC, PCS 完成数据处理后, 关闭 RX 解扰, FEC 等逻辑, 但 AMCTL 锁定电路需持续识别通道上的输入数据, 用于监控是否收到打开 FEC 和扰码的 AMCTL 控制命令。

#### 打开 PCS 处理逻辑流程:

1. PCS 检测到数据链路层有数据下发后, 停止发送 PRBS23 码流。
2. 停止发送 PRBS23 码流后立即发送一个 AMCTL, AMCTL CTRL\_TYPE 域设置为 FEC Control, AMCTL CTRL\_DETAIL 置为 Dynamic Open FEC。
3. 接收数据链路层数据, 进行 FEC 编码, 加扰等 PCS 处理。
4. 对端接收侧收到 AMCTL 并对内容进行解析, 如果 AMCTL CTRL\_TYPE == FEC Control 同时 AMCTL CTRL\_DETAIL == Dynamic Open FEC, 则打开 RX 解扰, FEC 译码等逻辑, 开始正常接收数据。

## 3.3 物理媒介适配子层

物理媒介适配子层从物理编码子层接收并行数据, 完成串行转换 ( Serializer ), 执行格雷编码 ( 针对 PAM4 调制 ) 和预编码功能后发送到 TX Lane 上; 从 RX Lane 上接收串行码流, 执行时钟数据恢复 ( Clock Data Recovery, CDR ), 格雷解码 ( 针对 PAM4 调制 ) 和预解码, 完成解串 ( Deserializer ) 后发送给物理编码子层。

PMA 支持对高速信号提供驱动及预加重, 均衡功能, 以补偿传输线的高频损耗及缓解高速信号的码间干扰 ( Inter-Symbol Interference, ISI ) 问题。

PMA 应提供时钟数据恢复功能, 从接收的串行码流中恢复时钟信号。

PMA 应支持物理编码子层通过介质无关的方式与多种物理介质链路连接, 支持每个端口的发送方向或者接收方向工作在 x1, x2, x4, x8 链路宽度, 每个端口的发送方向链路宽度可以和接收方向链路宽度相等, 也可以不相等。

PMA 应可以工作在 PHY Mode-1 或者 PHY Mode-2，工作模式应在上电前配置好。每种模式支持的速率及调制方式见 3.1.2 节。

当使用 PAM4 调制时，PMA 应提供格雷编码功能；同时 PMA 还应提供预编码功能。

### 3.3.1 格雷编码

当使用 PAM4 调制时，发往 PMA 的数据以 2 比特为粒度进行格雷编码。仅对需要加扰的 symbol 进行格雷编码。

PAM4 信号有四个电平，每个电平映射到 2 比特，从低到高为 00,01,10,11。

当采用 PAM4 调制时电平之间的容差变小，故而会因此产生相较 NRZ 调制时更高的误码发生概率。格雷编码的目的是当一个电平被接收器误判为临近电平时，也只会导致 1 比特错误。

下面是格雷编码映射表：

表 3-11 PAM4 编码和电平映射表

原始 PAM4 数据	格雷编码后数据	格雷编码后对应电平
00	00	0
01	01	1
11	10	2
10	11	3

PAM4 编码电压示意图如下：

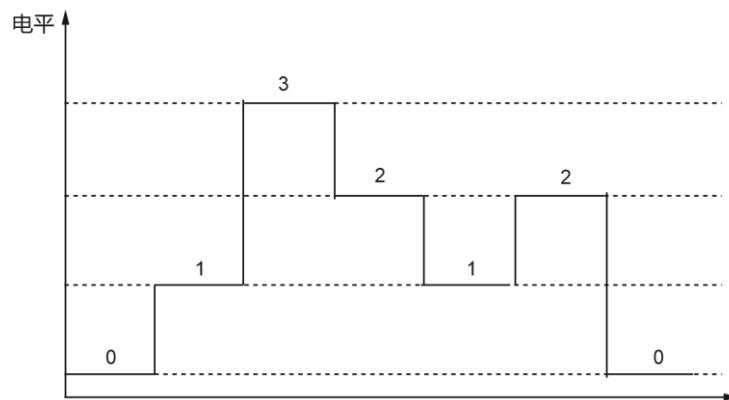


图 3-21 PAM4 电平示意图

### 3.3.2 预编码

UB 支持预编码功能用于消除突发信道下的错误传递。发送方向上每个通道对数据进行预编码处理，接收方向上每个通道对数据进行解码处理，预编码示意图如下：

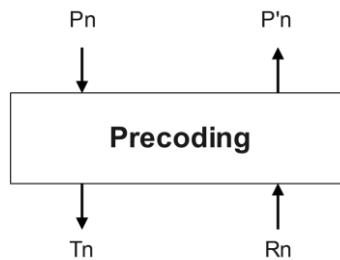


图 3-22 预编码示意图

当采用 NRZ 调制时：

- 在发送方向，输入 Precoding 数据是  $P_n$ ，输出数据是  $T_n$ ， $T_{n-1}$  代表 Precoding 前一个 UI ( Unit Interval ) 的输出， $T_n = (P_n \wedge T_{n-1})$ ，如果前一个 UI 的数据不加扰， $T_{n-1}$  需要置为 1'b1。
- 在接收方向，输入 Precoding 数据是  $R_n$ ，输出数据是  $P'_n$ ， $R_{n-1}$  代表 Precoding 前一个 UI 的输出， $P'_n = (R_n \wedge R_{n-1})$ ，如果前一个 UI 的数据不加扰， $R_{n-1}$  需要置为 1'b1。

当采用 PAM4 调制时：

- 在发送方向，输入 Precoding 数据是  $P_n$ ，输出数据是  $T_n$ ， $T_{n-1}$  代表 Precoding 前一个 UI 的输出， $T_n = (P_n - T_{n-1}) \bmod 4$ ，如果前一个 UI 的数据不加扰， $T_{n-1}$  需要置为 2'b00。
- 在接收方向，输入 Precoding 数据是  $R_n$ ，输出数据是  $P'_n$ ， $R_{n-1}$  代表 Precoding 前一个 UI 的输出， $P'_n = (R_n + R_{n-1}) \bmod 4$ ，如果前一个 UI 的数据不加扰， $R_{n-1}$  需要置为 2'b00。

仅对扰码的比特做 Precoding。

## 3.4 链路状态管理

链路状态管理功能通过链路管理状态机 ( Link Management State Machine, LMSM ) 来完成链路训练过程，将链路训练成可供数据链路层使用的状态；在链路出现异常情况时执行恢复流程，增强链路的可用性。

### 3.4.1 链路管理码

链路管理码 LMB ( Link Management Block ) 由物理层产生，用于支持链路训练和链路状态管理。每个 LMB 由 16 个 symbol 组成，每个 symbol 是 8 比特。

在一个由多条 Lane 组成的 Link 上，同一时刻所有 Lane 均应发送相同类型的 LMB。

LMB 包含如下两种类型：

- Link Training Block ( LTB )
- Exit Electrical Idle Block ( EEIB )

### 3.4.1.1 LTB

LTB ( Link Training Block ) 是 LMB 的一种。在链路训练期间，在一条 Link 的两端互相收发 LTB 进行链路训练。

本规范定义了 4 种类型的 LTB：

DLTB ( Discovery LTB ), CLTB ( Config LTB ), RLTB ( Retrain LTB ), ELTB ( Equalization LTB )。

如果当前 LTB 的 symbol0 ~ symbol11 等于前一个 LTB 的 symbol0 ~ symbol11，并且 CRC 校验通过，则认为前后两个 LTB 是连续的。

LTB 中的 Reserved bit 要求如下：

- TX 应将 Reserved bit 置为 0；
- RX 不能通过检测 Reserved bit 来判断 LTB 是否有效；
- RX 不能通过检测 Reserved bit 来决定执行任何的控制功能；
- LTB 的 symbol0 ~ symbol11 参与 CRC 计算且将 CRC 结果填入 LTB CRC[7:0] ( symbol 12 ) 和 LTB CRC[15:8] ( symbol 13 )；CRC 多项式如下：

$$x^{16} + x^{14} + x^{12} + x^{11} + x^8 + x^5 + x^4 + x^2 + 1$$

各种 LTB 的主要功能如下：

**DLTB**：用于在 Discovery 状态中进行符号锁定，交换链路初始化信息，确定 TX/RX Link Width；以及在固定速率模式下用于 RXEQ\_Optimize 状态中进行符号锁定及 RX 均衡适配。

**CLTB**：用于在 Config 状态中协商均衡模式、FEC 模式等。

**RLTB**：用于在 Retrain 状态中进行 Link 速率切换以及 Link 宽度调整，错误恢复等。

**ELTB**：用于在 Equalization 状态中进行链路均衡协商。

上述 LTB 格式及其域段详细定义如下面表 3-12~表 3-15 所示。

表 3-12 DLTB 定义表

Symbol 号	描述
0	<b>Type</b> 0xA0: Discovery.Active; 0xA1: Discovery.Confirm; 0xE0: RXEQ_Optimize; Others: Reserved。
1	<b>Link_ID</b> <b>Bits [7:0]</b> : 0-254, NULL; NULL 值是 8'hFF。
2	<b>Lane_ID</b> <b>Bits [7:0]</b> : 0-7, NULL; NULL 值是 8'hFF。

Symbol 号	描述
3	<p><b>TLW (TX Link Width)</b></p> <p><b>Bits [5:0]:</b></p> <ul style="list-style-type: none"> <li>6'b00_0001: x1</li> <li>6'b00_0010: x2</li> <li>6'b00_0100: x4</li> <li>6'b00_1000: x8</li> <li>Others: Reserved</li> </ul> <p>Bits [7:6]: Reserved。</p>
4	<p><b>RLW (RX Link Width)</b></p> <p><b>Bits [5:0]:</b></p> <ul style="list-style-type: none"> <li>6'b00_0001: x1</li> <li>6'b00_0010: x2</li> <li>6'b00_0100: x4</li> <li>6'b00_1000: x8</li> <li>Others: Reserved</li> </ul> <p>Bits [7:6]: Reserved。</p>
5	<p><b>Data_Rate_Support_1</b></p> <p>每个比特值为 1'b1 表示支持相应的速率等级，为 1'b0 表示不支持相应的速率等级。</p> <p><b>Bit 0:</b> Data Rate 0 (PHY Mode-1: 4.0 Gbps; PHY Mode-2: 2.578125 Gbps)。</p> <p><b>Bit 1:</b> Data Rate 1 (Reserved)。</p> <p><b>Bit 2:</b> Data Rate 2 (PHY Mode-2: 25.78125 Gbps)。</p> <p><b>Bit 3:</b> Data Rate 3 (PHY Mode-1: 自定义速率 1)。</p> <p><b>Bit 4:</b> Data Rate 4 (PHY Mode-2: 53.125 Gbps)。</p> <p><b>Bit 5:</b> Data Rate 5 (PHY Mode-1: 自定义速率 2)。</p> <p><b>Bit 6:</b> Data Rate 6 (Reserved)。</p> <p><b>Bit 7:</b> Data Rate 7 (PHY Mode-1: 自定义速率 3)。</p>
6	<p><b>Data_Rate_Support_2</b></p> <p><b>Bit 0:</b> Data Rate 8 (PHY Mode-2: 106.25 Gbps)。</p> <p>Bits [7:1]: Reserved。</p>
7	<p><b>FEC_Mode_Support</b></p> <p><b>Bit 0:</b> RS (128, 120, T=2)。</p> <ul style="list-style-type: none"> <li>1'b0: 不支持此 FEC 模式；</li> <li>1'b1: 支持此 FEC 模式。</li> </ul> <p><b>Bit 1:</b> RS (128, 120, T=4)。</p> <ul style="list-style-type: none"> <li>1'b0: 不支持此 FEC 模式；</li> <li>1'b1: 支持此 FEC 模式。</li> </ul>

Symbol 号	描述
	Bits [7:2]: Reserved。
8	<p><b>Bit 0: Port_Type</b>            1'b0: Secondary Port;            1'b1: Primary Port。</p> <p><b>Bits [7:1]: PortNego_Random_Value</b>            1-127, NULL;            NULL 值为 0。</p>
9	Reserved
10	<p><b>FEC_Interleave_Support</b>            每个比特值为 1'b1 表示相应的 Link Width 支持 FEC 交织模式，为 1'b0 表示不支持。</p> <p><b>Bit0:</b> x1  <b>Bit1:</b> x2  <b>Bit2:</b> x4  <b>Bit3:</b> x8</p> <p>Bits [7:4]: Reserved。</p>
11	<p><b>Bits [2:0]: RX_Lane0_ID</b>            链路中 Logic RX Lane0 的位置：            3'b000: Logic RX Lane0 在 Physical Lane0 上            3'b001: Logic RX Lane0 在 Physical Lane1 上            3'b011: Logic RX Lane0 在 Physical Lane3 上            3'b100: Logic RX Lane0 在 Physical Lane7 上            Others: Reserved。</p> <p><b>Bits [5:3]: MAX_LINK_WIDTH</b>            最大链路宽度：            3'b000: x1            3'b001: x2            3'b010: x4            3'b011: x8            Others: Reserved</p> <p>Bits [7:6]: Reserved。</p>
12	<p><b>Bits [7:0]: CRCL</b>            DLTB Symbol0-Symbol11 的 CRC[7:0]。</p>
13	<p><b>Bits [7:0]: CRCH</b>            DLTB Symbol0-Symbol11 的 CRC[15:8]。</p>
14	填充 0x5A

Symbol 号	描述
15	填充 0x5A

表 3-13 CLTB 定义表

Symbol 号	描述
0	<b>Type</b> 0xB0: Config.Active 0xB1: Config.Check 0xB2: Config.Confirm Others: Reserved
1	<b>Link_ID</b> <b>Bits [7:0]:</b> 0-254, NULL NULL 值为 8'hFF
2	<b>Lane_ID</b> <b>Bits [7:0]:</b> 0-7, NULL NULL 值为 8'hFF
3	<b>TLW (TX Link Width)</b> <b>Bits [5:0]:</b> 6'b00_0001: x1 6'b00_0010: x2 6'b00_0100: x4 6'b00_1000: x8 Others: Reserved <b>Bits [7:6]:</b> Reserved
4	<b>RLW (RX Link Width)</b> <b>Bits [5:0]:</b> 6'b00_0001: x1 6'b00_0010: x2 6'b00_0100: x4 6'b00_1000: x8 Others: Reserved <b>Bits [7:6]:</b> Reserved
5	Reserved
6	<b>EQ_Mode_Ctrl</b> <b>Bits [1:0]: EQ_Mode</b> 2'b00: Full_EQ 2'b01: Only_Highest_Data_Rate_EQ 2'b10: Skip_EQ

Symbol 号	描述
	<p>2'b11: Reserved 当变量 LinkUp==0 且支持 Data Rate 1 或者更高速率时，此域段用于 Config 状态进行均衡模式协商确定。</p> <p><b>Bits [4:2]: FEC_Mode_Ctrl</b> 3'b000: FEC Bypass 3'b001: RS ( 128,120,T=2 ) 3'b010: RS ( 128,120,T=4 ) Others: Reserved</p> <p><b>Bit 5: FEC_Interleave_Ctrl</b> 1'b1: Enable FEC Interleave 1'b0: Disable FEC Interleave</p> <p>Bits [7:6]: Reserved</p>
7	Bits [7:0]: Reserved
8	Bits [7:0]: Reserved
9	Bits [7:0]: Reserved
10	Bits [7:0]: Reserved
11	Bits [7:0]: Reserved
12	<p><b>Bits [7:0]: CRCL</b> CLTB Symbol0-Symbol11 的 CRC[7:0]</p>
13	<p><b>Bits [7:0]: CRCH</b> CLTB Symbol0-Symbol11 的 CRC[15:8]</p>
14	填充 0x5A
15	填充 0x5A

表 3-14 RLTB 定义表

Symbol 号	描述
0	<p><b>Type</b></p> <p>0xC0: Retrain.Active 0xC1: Retrain.Confirm 0xC2: Retrain.LP1_PHY_Up 0xC3: Retrain.EQ_Initial Others: Reserved</p>
1	Bits [7:0]: Reserved

Symbol 号	描述
2	<p><b>EQ_Ctrl1</b></p> <p><b>Bits [5:0]:</b> 当 RLTB.Type == Retrain.EQ_Initial 时，该域段为：  <b>Local_HS:</b> 在 Retrain.Confirm 状态中用于下一个速率的 Local_HS 值。          否则，该域段为：  <b>Post-Cursor:</b> 当前速率下的 Post-Cursor 系数值。</p> <p>Bits [7:6]: Reserved</p>
3	<p><b>EQ_Ctrl2</b></p> <p>当 RLTB.Type == Retrain.EQ_Initial 时：  <b>Bits [5:0]: Local_LS</b>          在 Retrain.Confirm 状态中传递用于下一个速率的 Local_LS 值。          Bits [7:6]: Reserved          否则：          如果当前速率的调制模式是 NRZ：  <b>Bits [5:0]: Pre-Cursor</b> 当前速率下的 Pre-Cursor 系数值。          Bits [7:6]: Reserved          如果当前速率的调制模式是 PAM4：  <b>Bits [3:0]: Second_Pre-Cursor</b> 当前速率下的 Second_Pre-Cursor 系数值。  <b>Bits [7:4]: Third_Pre-Cursor</b> 当前速率下的 Third_Pre-Cursor 系数值。</p>
4	<p><b>EQ_Ctrl3</b></p> <p>当 RLTB.Type == Retrain.EQ_Initial 时：  <b>Bits [3:0]: Remote_Initial_TX_Preset</b>          Bits [7:4]: Reserved          否则：  <b>Bits [7:0]: Cursor</b> 当前速率下的 Cursor 系数值。</p>
5	<p><b>Data_Rate_Support_1</b></p> <p>每个比特值为 1'b1 表示支持相应的速率等级，为 1'b0 表示不支持相应的速率等级。</p> <p><b>Bit 0:</b> Data Rate 0 (PHY Mode-1: 4.0 Gbps; PHY Mode-2: 2.578125 Gbps)  <b>Bit 1:</b> Data Rate 1 (Reserved)  <b>Bit 2:</b> Data Rate 2 (PHY Mode-2: 25.78125 Gbps)  <b>Bit 3:</b> Data Rate 3 (PHY Mode-1: 自定义速率 1)  <b>Bit 4:</b> Data Rate 4 (PHY Mode-2: 53.125 Gbps)  <b>Bit 5:</b> Data Rate 5 (PHY Mode-1: 自定义速率 2)  <b>Bit 6:</b> Data Rate 6 (Reserved)  <b>Bit 7:</b> Data Rate 7 (PHY Mode-1: 自定义速率 3)</p>

Symbol 号	描述
6	<p><b>Data_Rate_Support_2</b></p> <p>Bit 0: Data Rate 8 Supported (PHY Mode-2 106.25 Gbps)        Bits [6:1]: Reserved  <b>Bit 7: Change_Speed</b> 当链路想启动切速时有效（值为 1'b1）。</p>
7	<p><b>Bits [2:0]: FEC_Mode_Ctrl</b></p> <p>3'b000: FEC Bypass        3'b001: RS(128, 120, T=2)        3'b010: RS(128, 120, T=4)        Others: Reserved</p> <p><b>Bit 3: FEC/CRC_Mode_Reject</b></p> <p>1'b0: 不拒绝；        1'b1: 拒绝。</p> <p><b>Bits [5:4]: Pre-FEC_BER_Measurement_Status</b></p> <p>2'b00: 不测量 BER，或者使用软件配置的 FEC 模式；        2'b01: 开始测量 BER；        2'b10: 完成测量 BER；        2'b11: Reserved。</p> <p><b>Bit 6: FEC_Interleave_Enable</b></p> <p>1'b0: 不使能 FEC 交织；        1'b1: 使能 FEC 交织。</p> <p><b>Bit 7: FEC_Interleave_Ctrl_Reject</b></p> <p>1'b0: 不拒绝；        1'b1: 拒绝。</p>
8	Bits [7:0]: Reserved
9	<p><b>Link_Ctrl.</b></p> <p>Bits [6:0]: Reserved</p> <p><b>Bit 7: Request_Equalization</b></p> <p>1'b0: 无均衡请求；        1'b1: 有均衡请求。</p>
10	<p><b>Bits [2:0]: CRC_Mode_Ctrl</b></p> <p>3'b000: No CRC        3'b001: CRC30        Others: Reserved</p> <p>Bit [3]: Reserved</p> <p><b>Bits [7:4]: Local_TX_Preset</b></p> <p>当前速率下的 Local TX Preset 值。</p>

Symbol 号	描述
11	<p>如果当前速率的调制模式是 PAM4:</p> <p><b>Bits [5:0]: First_Pre-Cursor</b> 当前速率下的 First_Pre-Cursor 系数值。</p> <p>Bits [7:6]: Reserved</p> <p>否则:</p> <p>Bits [7:0]: Reserved</p>
12	<p><b>Bits [7:0]: CRCL</b></p> <p>RLTB Symbol0-Symbol11 的 CRC[7:0]</p>
13	<p><b>Bits [7:0]: CRCH</b></p> <p>RLTB Symbol0-Symbol11 的 CRC[15:8]</p>
14	填充 0x5A
15	填充 0x5A

表 3-15 ELTB 定义表

Symbol 号	描述
0	<p><b>Type</b></p> <p>0xD0: Equalization Others: Reserved</p>
1	<p>Bits [4:0]: Reserved</p> <p><b>Bit 5: EQ_Reject</b></p> <p>1'b0: 接受; 1'b1: 拒绝。</p> <p>Bits [7:6]: Reserved</p>
2	<p><b>EQ_Ctrl1</b></p> <p><b>Bits [1:0]: Current_EQ_Phase</b></p> <p>2'b00: Coarsetune_Active 2'b01: Coarsetune_Confirm 2'b10: Secondary Port: EQ.Active/Primary Port: EQ.Passive 2'b11: Secondary Port: EQ.Passive/Primary Port: EQ.Active</p> <p>Bit 2: Reserved</p> <p><b>Bit 3: Preset_Mode_En</b></p> <p>1'b0: 不使用 Preset 方式; 1'b1: 使用 Preset 方式。</p> <p><b>Bits [7:4]: TX_Preset</b></p>

Symbol 号	描述
3	<p><b>EQ_Ctrl2</b> 仅当 Preset_Mode_En==0 时有效。 如果当前速率的调制模式是 NRZ:     <b>Bits [5:0]: Pre-Cursor</b> 当前速率下的 Pre-Cursor 系数值。     Bits [7:6]: Reserved。 如果当前速率的调制模式是 PAM4:     <b>Bits [3:0]: Second_Pre-Cursor</b> 当前速率下的 Second_Pre-Cursor 系数值。     <b>Bits [7:4]: Third_Pre-Cursor</b> 当前速率下的 Third_Pre-Cursor 系数值。</p>
4	<p><b>EQ_Ctrl3</b> 仅当 Preset_Mode_En==0 时有效。 <b>Bits [5:0]: Post-Cursor</b> 当前速率下的 Post-Cursor 系数值。 Bits [7:6]: Reserved。</p>
5	<p><b>EQ_Ctrl4</b> 仅当 Preset_Mode_En==0 时有效。 <b>Bits [7:0]: Cursor</b> 当前速率下的 Cursor 系数值。</p>
6	<p>如果当前速率的调制模式是 PAM4:     <b>Bits [5:0]: First_Pre-Cursor</b> 当前速率下的 First_Pre-Cursor 系数值。     Bits [7:6]: Reserved。 否则:     Bits [7:0]: Reserved。</p>
7-11	Reserved。
12	<p><b>Bits [7:0]: CRCL</b> ELTB Symbol0-Symbol11 的 CRC[7:0]。</p>
13	<p><b>Bits [7:0]: CRCH</b> ELTB Symbol0-Symbol11 的 CRC[15:8]。</p>
14	填充 0x5A
15	填充 0x5A

### 3.4.1.2 EEIB

EEIB ( Exit Electrical Idle Block ) 是一种低频码型，以确保接收器电气空闲退出检测电路能够检测到电气空闲退出信号从而完成电气空闲状态退出。

一个 EEIB 序列在不同速率下包括的 EEIB 个数不一样：

速率小于或等于 4.0 Gbps 时包括 1 个 EEIB；

速率为 25.78125 Gbps 时包括 2 个 EEIB；

速率为 53.125 Gbps 时包括 4 个 EEIB；

速率为 106.25 Gbps 时包括 8 个 EEIB。

一个 EEIB 序列中的 EEIB 背对背不中断传输才能被认为是一个 EEIB 序列。

**表 3-16 2.578125/4.0 Gbps 的 EEIB 序列 ( NRZ )**

Symbol Number	Value	Description
0, 2, 4, 6, 8, 10, 12, 14	0x00	此 EEIB 是由 8 个 0 和 8 个 1 交织的低频信号。
1, 3, 5, 7, 9, 11, 13, 15	0xFF	

**表 3-17 25.78125 Gbps 的 EEIB 序列 ( NRZ )**

Symbol Number	Value	Description
0, 1, 2, 3, 8, 9, 10, 11	0x00	此 EEIB 是由 32 个 0 和 32 个 1 交织的低频信号。
4, 5, 6, 7, 12, 13, 14, 15	0xFF	

**表 3-18 53.125 Gbps 的 EEIB 序列 ( PAM4 )**

Symbol Number	Value	Description
0, 1, 2, 3, 4, 5, 6, 7	0x00	32 个 UI 电平 0
8, 9, 10, 11, 12, 13, 14, 15	0xFF	32 个 UI 电平 3

**表 3-19 106.25 Gbps 的 EEIB 序列 ( PAM4 )**

Symbol Number	Value	Description
0-15	0x00	64 个 UI 电平 0
16-31	0xFF	64 个 UI 电平 3

注：自定义速率的 EEIB 序列由厂商自定义。

### 3.4.2 链路状态管理特性

链路状态管理功能支持以下特性：

- 电链路和光链路
- FEC 模式协商，支持动态 FEC 模式切换

- 链路宽度协商，支持动态链路宽度切换
- 故障时强制快速降通道
- 数据速率协商和速率切换
- Lane 顺序翻转
- Lane 极性检测和翻转
- 均衡协商

### 3.4.2.1 端口类型协商

一条 UB Link 连接的一个端口是 Primary Port，另一个端口是 Secondary Port，两者在链路训练过程中的角色不同。

如果软件使能了端口类型协商特性，链路训练的 Discovery 状态期间两个 Port 会通过端口类型协商机制确定 Primary Port 和 Secondary Port。

每个 Port 中有一个随机数产生器用于在端口类型协商过程中通过产生随机数来竞争成为 Primary Port。

`DLTB.Port_Type` (`Symbol8.bit0`) 代表本 Port 的当前类型，`DLTB.PortNego_Random_Value` (`Symbol8.bits[7:1]`) 表示本 Port 的随机数值。

随机数值大的 Port 为 Primary Port。

在 Discovery 状态期间 Link 两端通过端口类型协商确定 Port 类型的流程如下：

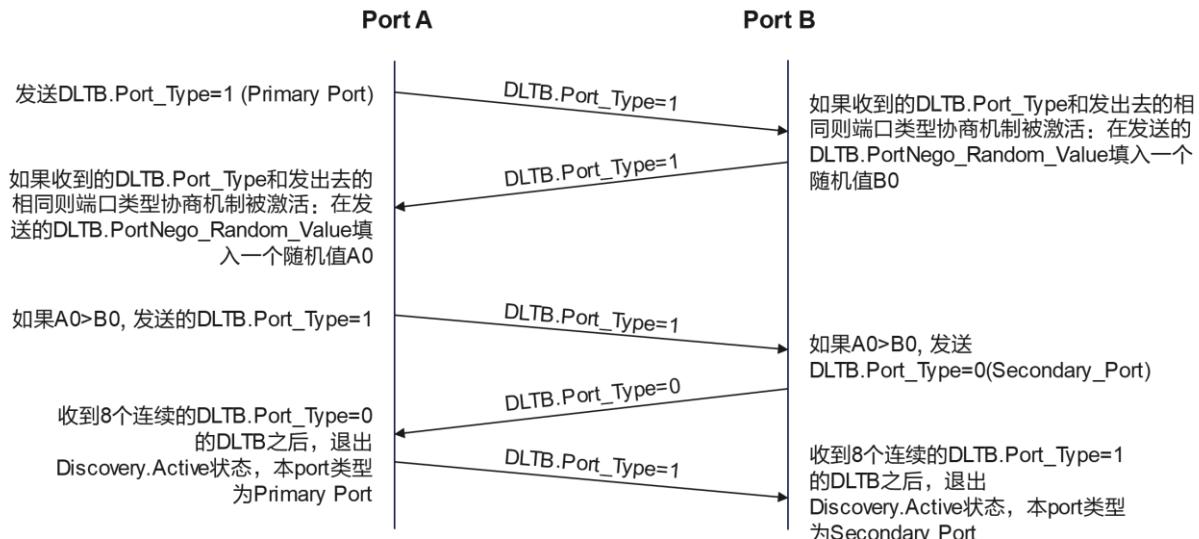


图 3-23 端口类型协商流程图

如果一端收到的 DLTB 中的随机数等于本端发出去的随机数，则产生一个新的随机数再发送出去。以后只有当接收到的 DLTB 中的随机数值不等于之前接收的 DLTB 中的随机数值时才进行比较。

如果没有使能端口类型协商特性，软件配置 Link 两端的 Port 类型之后，两个 Port 就可以依据配置好的 Port 类型进行链路训练。

### 3.4.2.2 链路宽度协商

一条 UB Link 由一条或者多条 Lanes 组成，Lane 的数量即为 Link Width。UB 规范支持的 Link Width 是 x1, x2, x4, x8，这些是标准的 Link Width。

当 TX 发送 LMB 时，一条 Link 上所有的 Lane 在同一时刻发送的 LMB 类型相同。

UB 规范支持对称链路 (symmetric Link)，即一个 Port 的 TX 和 RX 的 Lane 的数量是相等的。在链路训练开始之前链路两端 Port 支持的 Lane 数可能不同，在 Discovery 状态之后，将得到协商好的最大 Link Width 来组成对称链路。

UB 也支持两个具有不同的 TX 和 RX Lane 数量的 Port 来建立不对称链路 (Asymmetrical Link)。不对称链路在每一个方向的 Link Width 应是标准 Link Width。例如，PortA TX 是 x8, RX 是 x4；PortB TX 是 x4, RX 是 x8；在链路训练完成后，组成的链路将是：从 A 到 B 方向的 Link Width 是 x8，而从 B 到 A 方向的 Link Width 是 x4。

### 3.4.2.3 快速动态链路宽度切换

链路宽度可以通过链路管理状态机来改变，状态流程跳变如下：

Link\_Active->Retrain->Discovery->Config->Send\_NullBlock->Link\_Active

在这种方法中 Link 跳进了训练状态从而会中断业务数据，这种途径通常在链路异常需要重新协商链路宽度时使用。

UB 也支持快速动态链路宽度切换 (Quickly Dynamic Link Width Switch, QDLWS) 特性，在不中断业务数据的情况下进行链路宽度切换。

在 QDLWS 被使能的情况下，Primary Port 可以请求增加或减少 Lane 数量的链路宽度切换，Secondary Port 可以接受或者拒绝这样的切换请求。

QDLWS 需要遵守下列规则：

- Link 首先在最大的链路宽度上完成相应速率的均衡协商过程，并训练到 Link\_Active 状态。
- 新的链路带宽调整请求在前一个调整请求完成至少 1us 之后发出。
- 请求的链路宽度是标准的链路宽度。
- 一个 Port 可以独立请求调整 TX 宽度或者 RX 宽度，或者同时请求调整 TX 和 RX 宽度。
- 收到 QDLWS 请求的 Port 如果没有准备好，可以反馈 NAK 来拒绝请求。

QDLWS 流程如下：

### 示例 1：从 TX4RX2 到 TX2RX2 的链路宽度减少流程

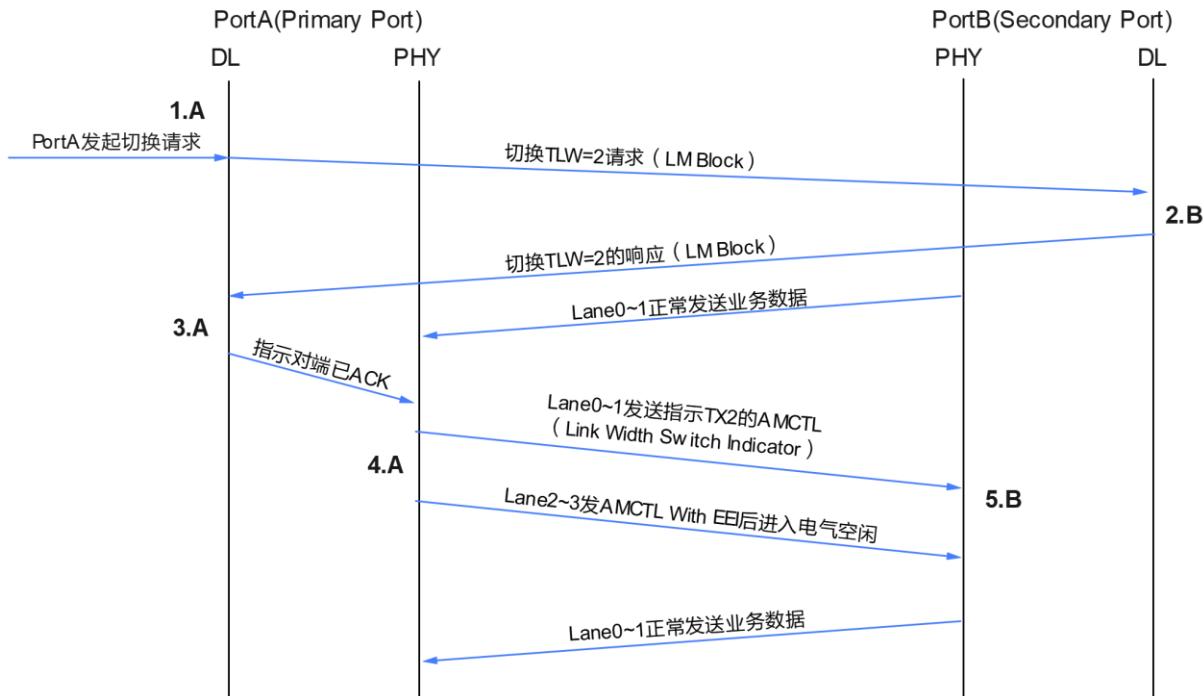


图 3-24 QDLWS 链路宽度减少流程图

PortA 发起降 Lane 请求，请求本端 TX 宽度从 x4 切换到 x2。

**1.A:** PortA 的数据链路层发送动态升降 Lane 的数据链路层控制块 LM Block (见 4.3.3.7 节)，LM Block 的 Byte0.Bit0 = 1' b1 (Change TLW request)，Byte2.Bits[5:0] = 6' b000010 (TX Link Width = x2)，向 PortB 发起降 TX Lane 到 x2 的请求。

**2.B:** PortB 的数据链路层收到 PortA 发送的 LM Block，解析各个域段，确认当前可以进行降 Lane 动作，则向 PortA 发送响应 LM Block，LM Block 的 Byte1.Bit0 = 1' b1 (Change TLW request RSP = ACK)，Byte3.Bits[5:0] = 6' b000010 (RX Link Width = x2)，确认接受 PortA 的降 Lane 请求。

若 PortB 没有准备好，或者不接受 PortA 的降 Lane 请求，则向 PortA 发送响应 LM Block，LM Block 的 Byte1.Bit0 = 1' b0 (Change TLW request RSP = NAK)，Byte3.Bits[5:0] = 6' b000100 (RX Link Width = x4)，拒绝 PortA 的降 TX Lane 请求。

**3.A:** PortA 的数据链路层收到 PortB 发送的 LM Block，解析各个域段，确认 PortB 已返回 ACK。

若 PortA 接收到的 LM Block 里 PortB 返回的是 NAK，流程结束。

**4.A:** PortA 的数据链路层指示物理层发起降 Lane 过程，物理层在 Lane0-Lane1 上发送指示 TX2 的控制 AMCTL (AMCTL CTRL\_TYPE = Link Width Switch Indicator, AMCTL CTRL\_DETAIL = x2)，之后的数据按 x2 发送；同时在 Lane2-Lane3 上发送 AMCTL with EEI，之后 Lane2-Lane3 进入电气空闲状态，关闭共模电压等 SerDes 的电路。

**5.B:** PortB 的物理层收到 PortA 发送指示 TX2 的 AMCTL，之后的数据按 x2 进行 Deskew 和解包处理，链路即完成 x4->x2 的切换过程。

### 示例 2：从 TX2RX4 到 TX4RX4 的链路宽度增加流程

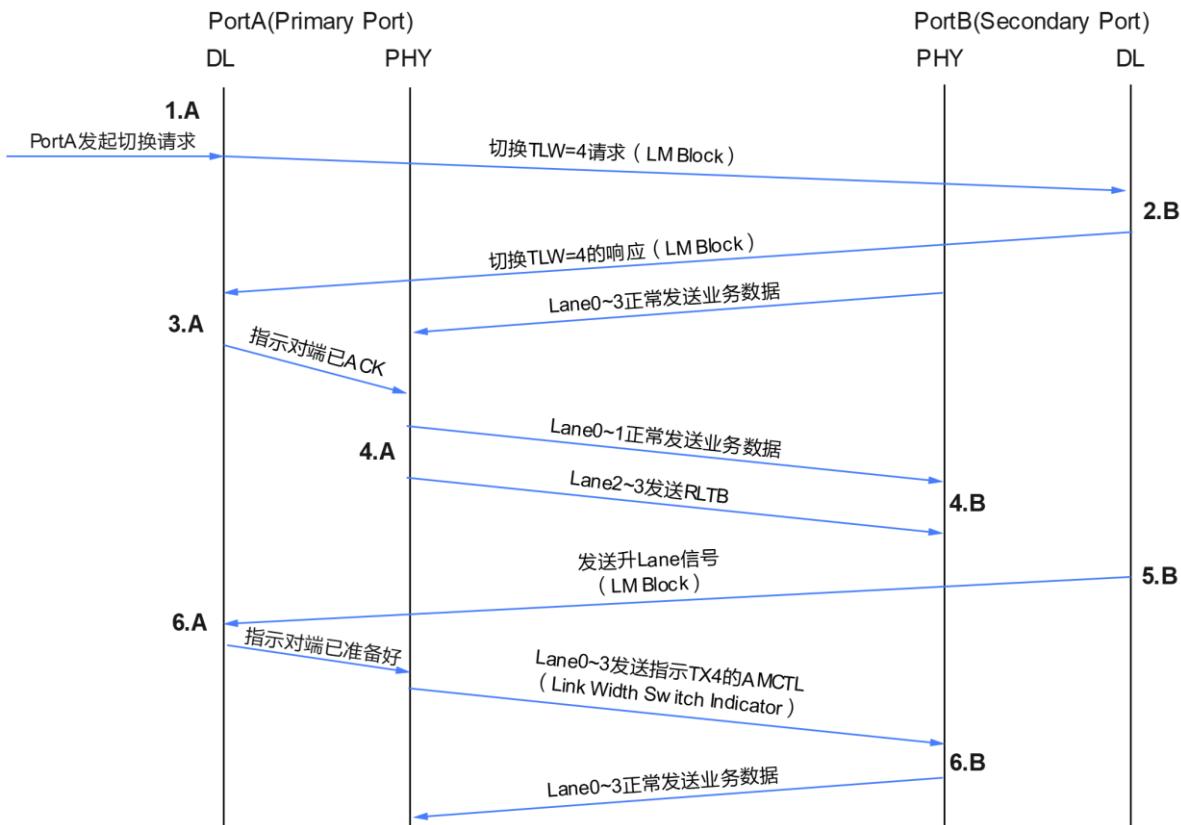


图 3-25 QDLWS 链路宽度增加流程图

PortA 发起升 Lane 请求，请求本端 TX 宽度从 x2 切换到 x4。

**1.A:** PortA 的数据链路层发送 LM Block，LM Block 的 Byte0.Bit0 = 1' b1 ( Change TLW request )，Byte2.Bits[5:0] = 6' b000100 ( TX Link Width = x4 )，向 PortB 发起升 TX Lane 到 x4 的请求。

**2.B:** PortB 的数据链路层收到 PortA 发送的 LM Block，解析各个域段，确认当前可以进行升 Lane 动作，则向 PortA 发送响应 LM Block，LM Block 的 Byte1.Bit0 = 1' b1( Change TLW request RSP = ACK )，Byte3.Bits[5:0] = 6' b000100 ( RX Link Width = x4 )，确认接受 PortA 的升 Lane 请求；同时指示本端物理层的 Lane2-Lane3 退出电气空闲状态，开启 CDR，DFE 等电路，准备接收数据。

若 PortB 没有准备好，或者不允许 PortA 的升 Lane 请求，则向 PortA 发送响应 LM Block，LM Block 的 Byte1.Bit0 = 1' b0 ( Change TLW request RSP = NAK )，Byte3.Bits[5:0] = 6' b000100 ( RX Link Width = x2 )，拒绝 PortA 的升 TX Lane 请求。

**3.A:** PortA 的数据链路层收到 PortB 发送的 LM Block，解析各个域段，确认 PortB 已返回 ACK。

若 PortA 接收到的 LM Block 里 PortB 返回的是 NAK，流程结束。

**4.A:** PortA 的数据链路层指示本端物理层发起升 Lane 过程，物理层在 Lane0-Lane1 上继续发送当前数据，同时指示物理层的 Lane2-Lane3 退出电气空闲状态，开启发送电路。Lane2-Lane3 完成电气空

闲退出后，即开始连续不断的发送 RLTB 码流，除了 RLTB.Type == Retrain.LP1\_PHY\_Up 之外，RLTB 的其它域段值和 Retrain.Active 状态中 RLTB 域段值一样（非切速，非 EQ request）。在发送 RLTB 前，需将 AMCTL 插入周期协商切换为短周期，其周期值可以配置，默认值为 640 个 symbol 插入一次(配置值需保证一个连续的 RLTB 不被打断)。在升 Lane 期间，发送数据的 Lane 和发送 RLTB 的 Lane，需按配置的短周期定期插入 AMCTL。

**4.B:** PortB 接收端在 Lane2-Lane3 上持续检测 RLTB 码流，并校验 RLTB 的 CRC 是否正确，如果可以收到连续 8 个 CRC 正确的 RLTB，则数据链路层向对端发送 LM Block，指示 PortB 当前 Lane2-Lane3 已经可以正常工作；如果超过一定的时间 T0(T0==2ms)仍然无法收到连续 8 个 CRC 正确的 RLTB，则控制物理层启动 RX 自适应，调节 RX 的参数，并持续检测 RLTB，如果超过 T1(T1==22ms)时间仍然无法收到，则此次升 Lane 失败，流程结束。

**5.B:** PortB 数据链路层向对端发送 LM Block，LM Block 的 Byte1.Bit2 = 1' b1 (RX Lane up)，指示 PortB 当前 Lane2-Lane3 已经可以正常工作。

**6.A:** PortA 的数据链路层收到 PortB 的 LM Block，LM Block 信息指示 PortB 的 RX Lane2-Lane3 已经准备好，则 PortA 在 Lane0-Lane3 上发送指示 TX4 的控制 AMCTL (AMCTL CTRL\_TYPE = Link Width Switch Indicator, AMCTL CTRL\_DETAIL = x4)，之后的数据按 x4 进行发送。

**6.B:** PortB 的物理层收到 PortA 发送指示 TX4 的 AMCTL，之后的数据按 x4 进行 Deskew 和解包处理，链路即完成 x2->x4 的切换过程。在完成升 Lane 后将 AMCTL 插入周期协商切换为升 Lane 前的周期值。

注：当 Port 发起快速动态升降 Lane 时，需统计从向本端数据链路层发起升降 Lane 请求，到对端通过 LM Block 返回响应的时间，如果时间超过 T0+T1 ( 24ms )，则取消本次升降 Lane 请求，回到发起升降 Lane 前的状态。

#### 3.4.2.4 强制快速降通道

UB 支持在 Link\_Active 状态快速降 Lane，降 Lane 过程通过 AMCTL 的 Remote TX Link Width Switch Indicator 命令完成，LMSM 不用进入 Retrain 状态，可减少降 Lane 过程所需的时间，使用此机制时需满足以下规则：

1. 在检测到本端 RX Lane 异常时，至少 RX Lane0 是正常的才能走本流程；如果 RX Lane0 异常，则需要通过跳转到 Retrain 状态来执行降 Lane 过程。
2. 发起强制降 Lane 请求侧，其 RX 方向必须无 Lane reversal。
3. 接收强制降 Lane 请求侧，在满足上述规则情况下，无需返回 ACK 或者 NAK，直接发起降 Lane。
4. 发起侧 RX 通过异常 Lane 判断目标 Link Width 时，必须从 Lane0 开始确定 Link Width，同时 Link Width 必须为标准的 Link Width。

以 PortB RX Lane3 损坏为例，降 Lane 流程如下：

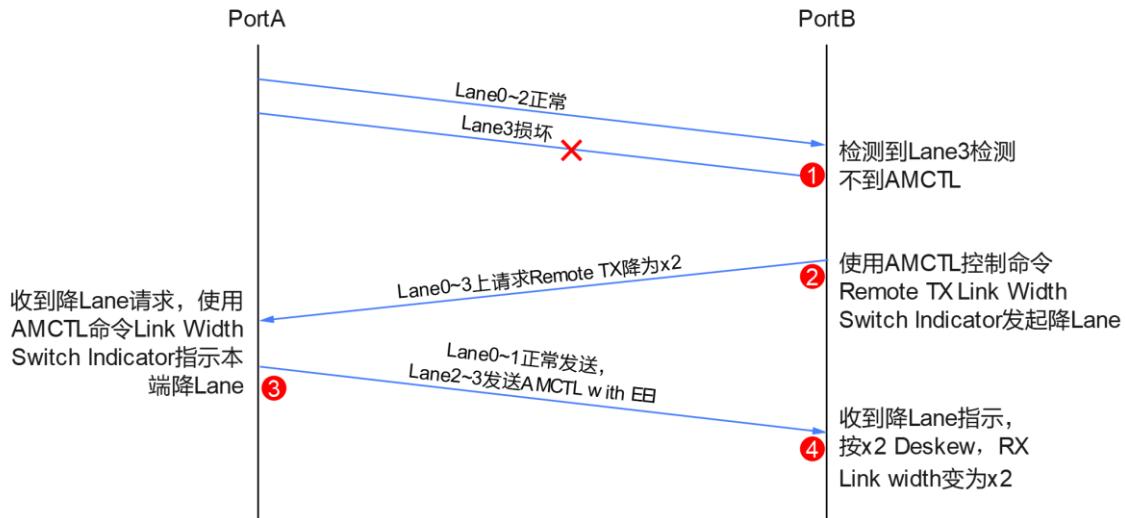


图 3-26 Link\_Active 下快速降 Lane 流程图

步骤 1 PortB 接收侧周期性检测各 Lane 是否收到 AMCTL，如果部分 Lane(非 Lane0)连续 N 次 ( $0 < N < 255$ , 用户自定义配置, 默认为 3)无法正确检测到 AMCTL, 则触发快速降 Lane 流程。

步骤 2 PortB 在所有激活的 Lane 上使用 AMCTL 向对端发起快速降 Lane 流程 , AMCTL.CTRL\_TYPE 置为 Remote TX Link Width Switch Indicator,AMCTL.CTRL\_DETAIL 置为目标 Link Width。

步骤 3 PortA 在任意激活的 Lane 上收到携带 Remote TX Link Width Switch Indicator 的 AMCTL,, 则根据 AMCTL.CTRL\_DETAIL 指示, 降低本端 TX Lane 宽度, 以降 Lane 到 x2 为例, 在 Lane0-Lane1 上发送 AMCTL, AMCTL.CTRL\_TYPE==Link Width Switch Indicator, AMCTL.CTRL\_DETAIL==x2; 在 Lane2-Lane3 上发送 AMCTL with EEI 后, Lane2-Lane3 进入电气空闲。

步骤 4 PortB 收到 AMCTL 中携带的降 Lane 指示后, PortB 的 Des skew 按 AMCTL 中指示的目标宽度执行, 按照 x2 接收数据, 流程结束。

### 3.4.2.5 数据速率协商

UB Link 支持速率协商。

链路使用 Data Rate 0 作为初始数据速率开始链路训练, Port 在 Discovery 状态期间通过 DLTB.Data\_Rate\_Support\_1 及 DLTB.Data\_Rate\_Support\_2 ( symbol 5 和 symbol 6 ) 广播其全部支持的速率, Link 的两端 Port 通过 LMSM 进行速率协商, 并在 Retrain 状态之后改变到两端广播都支持的共同最高新速率。如果新速率协商失败, LMSM 将切换回旧速率。

对于工作在光链路模式的链路, 在链路训练期间及训练完成后速率固定不变, 从始至终保持目标链路速率。例如一个光链路模式的链路目标速率是 53.125 Gbps, 链路训练从始至终都使用 53.125 Gbps 速率。

一个 Port 只支持 TX 和 RX 工作在同一个速率, 不支持 TX 和 RX 工作在不同速率。

### 3.4.2.6 通道极性翻转

为方便板级布线, UB 支持 Lane 极性翻转, 即一条 Lane 的 TX 正极允许连接到对端 RX 对应 Lane 的负极, TX 负极连接到对端 RX Lane 的正极。在链路训练的 Discovery 状态进行 Lane 翻转检测和调整。AMCTL 被用作 Lane 翻转的检测指示符号。

TX 边每条 Lane 上发送的 AMCTL 不翻转, 如果对端 RX 检测到某条 Lane 上 AMCTL 被翻转了, 则 RX 将该 Lane 的正极和负极反过来使用, 即 RX 负责 Lane 翻转的检测和调整。

在 PAM4 模式, 执行 Gray 解码之前完成 Lane 极性翻转。翻转时每一个 bit 直接翻转, 如下表所示:

表 3-20 PAM4 Lane 极性翻转示例

TX 原始数据	TX Gray 编码后	TX 电平	RX 电平	RX 原始数据	RX 翻转数据	RX Gray 解码后数据
00	00	0	3	11	00	00
01	01	1	2	10	01	01
10	11	3	0	00	11	10
11	10	2	1	01	10	11

### 3.4.2.7 通道顺序翻转

通常情况下一个 Port 的 TX Lane0 连接到对端的 RX Lane0, 它的 RX Lane0 连接到对端的 TX Lane0。

有些情况下为了板级布线方便, 两个 Port 之间的 Lane 连接顺序可以被翻转。比如对于一个 x4 Link, PortA 的 TX0, TX1, TX2, TX3 连接到 PortB 的 RX3, RX2, RX1, RX0; PortA 的 RX0, RX1, RX2, RX3 连接到 PortB 的 TX3, TX2, TX1, TX0。

Port 在 Probe 状态期间检测及调整 Lane0 的位置。详情见 3.4.3.2 节。

Primary Port 和 Secondary Port 都应支持 Lane 顺序翻转特性。

### 3.4.2.8 FEC 模式/CRC 模式切换

Port 在 Link\_Active 状态允许测量 Pre-FEC BER 并通过进入 Retrain 状态进行 FEC&CRC 模式协商和切换, Port 可以选择使用一种匹配当前 Pre-FEC BER 相应水平的 FEC 模式和 CRC 模式。

如果某个 Port 检测到当前 FEC/CRC 模式和当前 BER 不匹配, 希望切换到一个匹配的模式, 它可以发起 FEC/CRC 模式切换请求。对端 Port 可以根据自己的测量和判断决定接受切换请求或者拒绝请求。

FEC 模式和 CRC 模式切换参考流程见下图:

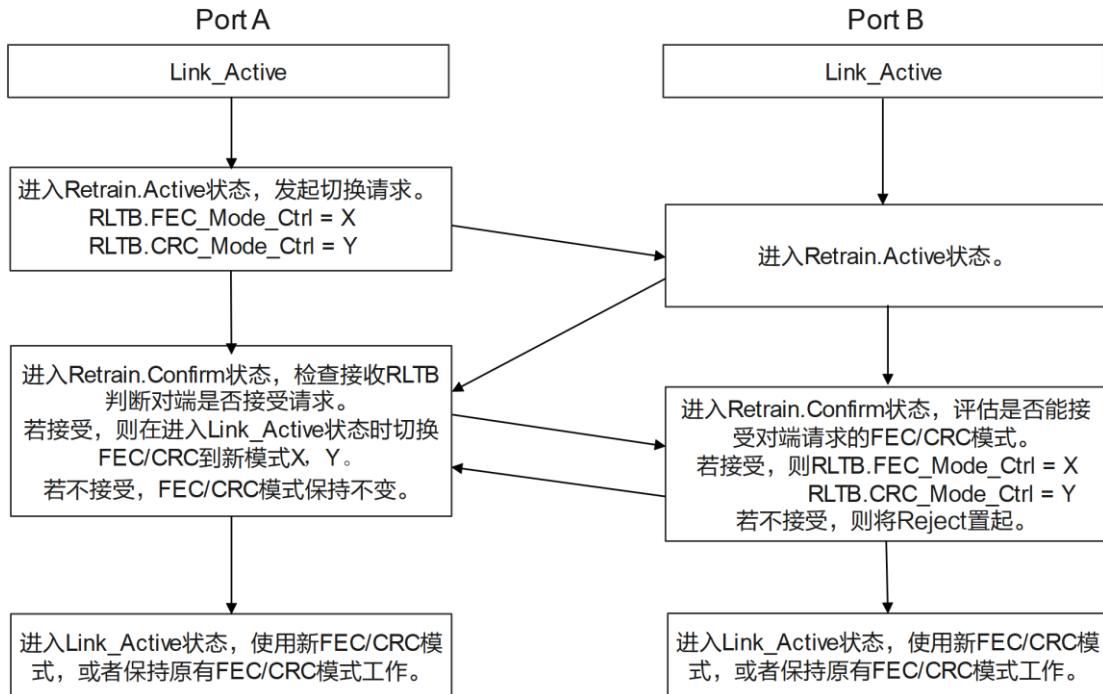


图 3-27 FEC/CRC 模式切换协商流程图

注 1: Port 允许只切换 FEC 模式, CRC 模式或者同时切换 FEC 及 CRC 模式。

注 2: 在切换后数据链路层需要确保两端的流控交互, 及 BCRC 的产生及验证是正确的。

注 3: Port 使用携带在 RLTB 中的 FEC 模式/CRC 模式域段进行切换协商。

注 4: 软件允许通过更改配置空间中 PHY Link Control 6,7,8,9 寄存器中相应速率下的 FEC 模式/CRC 模式, 然后写 PHY Control 1 寄存器中的 Retrain Link 比特来进行 FEC/CRC 模式切换。在进入 Retrain 状态之后的流程和上图相同。

### 3.4.2.9 链路均衡

对于 Data Rate 1 及以上速率的链路来说, 链路均衡机制调整 SerDes TX 和 RX 均衡参数来改善链路信号质量。

一个 Link 中全部的 Lane 都应参与均衡过程。当链路切换到 Data Rate 1 及以上速率时需要执行均衡过程, 除非这个链路两端的 Port 都声明不需要均衡过程。

一个 Port 的 TX 和 RX 均应在均衡过程结束后存储协商好的均衡参数, 以便当再次切换到此速率时使用这些均衡参数。

UB 支持如下均衡模式, 均衡模式在 Config 状态期间协商。

#### **Skip\_EQ 模式:**

在此模式链路从初始速率直接切换到两端所支持的最高速率, 链路管理状态机不会在任何速率执行均衡过程。此模式用于以前执行过 EQ 过程的 Link, 两端存储有协商好的均衡参数, 节省链路训练时间。

#### **Only\_Highest\_Data\_Rate\_EQ 模式:**

在此模式链路从初始速率直接切换到两端所支持的最高速率, 并且只在此最高速率上执行均衡过程。

例如两端支持的速率为 2.578125, 25.78125, 53.125, 106.25 Gbps, LMSM 将直接从 2.578125 Gbps 切换到 106.25 Gbps，并在 106.25 Gbps 速率上执行均衡过程。

#### Full\_EQ 模式：

在此模式链路应逐级切换直到两端所支持的最高速率，并在每一个 Data Rate 1 ( 包含 ) 以上速率执行均衡过程。例如两端共同支持的速率是 2.578125, 25.78125, 53.125, 106.25 Gbps，则速率逐级从 2.578125 -> 25.78125 -> 53.125 -> 106.25 Gbps 跳变，并在 25.78125, 53.125, 106.25 Gbps 速率上执行均衡过程。

Link 处于光链路模式时仅支持 Skip\_EQ 模式，且均衡模式在链路训练开始之前配置好。

如果是不对称链路，TX 可以使用固定均衡参数，接收端在 Equalization 状态执行均衡参数适配过程。

### 3.4.3 链路管理状态机

两个 Port 之间的一条或者多条 Lane 组成一个链路 ( Link ), 在完成链路训练之前，链路两端的 UBPU 不允许发送任何业务数据 ( Flit )。本节定义链路管理状态机，这是一种控制链路训练的硬件机制，通过链路训练，确定链路宽度，速率，FEC 模式，Lane 顺序翻转( Lane reversal )及 Lane 极性翻转( Lane polarity ) 等，将链路训练到能够进行业务数据收发的状态。

链路两端的每一个 Port 都有一个链路管理状态机，在电源及时钟信号稳定后，复位信号释放或者软件启动链路管理状态机开始链路训练。

链路管理状态机支持电链路，也支持光链路的链路训练。

链路管理状态机的状态跳转图如下所示：

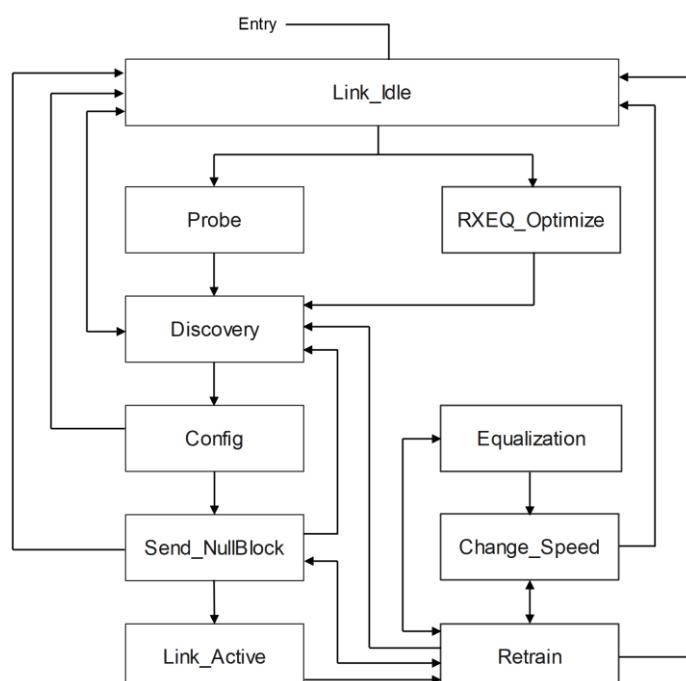


图 3-28 UB LMSM 状态跳转图

UB LMSM 状态说明如下。

### 3.4.3.1 Link\_Idle

- 退出复位后 LMSM 进入 Link\_Idle 状态，在此状态全部的 TX Lanes 处于电气空闲状态。
- LMSM 在此状态依据两个 Port 之间的连接类型来决定是否 bypass Probe 状态：
  - AC 耦合模式：TX 发出检测脉冲，通过 RC 电路 (resistor-capacitor circuit) 来检测对端接收器是否存在。此模式需要进入 Probe 状态。
  - 光链路模式：这种模式不发出检测脉冲来检测对端接收器是否存在，此模式 bypass Probe 状态。
- 在此状态变量初始化如下：

光链路模式下变量 *over\_fibre\_optical\_enable* 置为 1；其它情况下置为 0。

光链路模式下或者链路不支持速率切换时，变量 *fix\_data\_rate\_mode* 置为 1；否则置为 0。

光链路模式下变量 *bypass\_equalization* 置为 1；其它情况下置为 0。

光链路模式下，或者无法或不需要支持 Probe 时，变量 *bypass\_Probe* 置为 1；其它情况下置为 0。

变量 *Tx\_M* 等于寄存器 PHY Link Control 1 中 Target TX Link Width 域段的值。

变量 *Rx\_N* 等于寄存器 PHY Link Control 1 中 Target RX Link Width 域段的值。

注：在 PHY Mode-2 时，*Tx\_M* 等于 *Rx\_N*。

变量 *LinkUp* = 0；

变量 *LinkReady* = 0；

变量 *change\_speed* = 0；

变量 *start\_eq\_init* = 0；

变量 *symmetry\_link* = 0；

变量 *Retrain\_num* = 0；

当不使用 Skip\_EQ 模式时（即需要执行 EQ 协商过程），变量 *eq\_complete\_data\_rate1* - *eq\_complete\_data\_rate2* 复位为 0；

变量 *rx\_lane0\_id* = 0；

变量 *tx\_lane0\_id* = 0；

准备激活的 TX Lane 编号是：Lane Tx\_0 到 Lane Tx\_M-1；

准备激活的 RX Lane 编号是：Lane Rx\_0 到 Lane Rx\_N-1；

- 如果变量 *bypass\_Probe* == 0 且 PHY ready 且上层指示进入，则下一个状态是：Probe。
- 如果变量 *fix\_data\_rate\_mode* == 1 且 PHY ready 且上层指示进入，则下一个状态是：RXEQ\_Optimize。
- 如果变量 *bypass\_Probe* == 1 且 *fix\_data\_rate\_mode* == 0 且 PHY ready 且上层指示进入，则下一个状态是：Discovery。

“上层指示进入”意味着使用实现特定的方法来启动 LMSM 开始工作。

### 3.4.3.2 Probe

Probe 状态的目的是探测每条 Lane 对端的接收器是否存在, 确定初始的链路宽度以及 Lane0 的位置。处于 Probe 状态时, Port 通过每条 Lane 的 TX 发出检测脉冲, 依赖于接收端是否有端接, 电路 RC 值会不一样, TX 中的检测电路就可以通过检测返回波形来判断对端 RX Lane 是否存在端接。根据对端 RX 端接检测结果, 可以初步确定每个方向的链路宽度, 并确定 Lane0 的位置。

Probe 的子状态描述如下:

#### Probe.Wait

- 所有准备激活的 TX Lane 均处于电气空闲状态。
- 所有准备激活的 RX Lane 打开对地端接电阻。
- 链路训练的起始速率是 Data Rate 0。如果进入 Probe.Wait 时的速率不是 Data Rate 0, LMSM 应该保持在此子状态一定的时间 (时间值由实现决定), 并在此期间改变速率到 Data Rate 0。
- 如果满足下面任一条件, 则下一个状态是 Probe.Confirm:
  - 在任何准备激活的 RX Lane 上面检测到电气空闲退出。
  - 超时 (超时值由实现决定)。

#### Probe.Confirm

- 进入此状态后, TX 开始在所有准备激活的 TX Lane 上发送检测脉冲(在正极和负极上都要发送), 以检测对端接收侧是否有端接电阻存在。检测结果可用于决定是否做 Lane reversal。
- 如果在全部准备激活的 TX Lane 上检测到了对端端接, 则下一状态是 Discovery。
- 如果在任何准备激活的 TX Lane 上都没有检测到对端端接, 则下一状态是 Probe.Wait。
- 如果至少一条, 但不是全部准备激活的 TX Lane 上检测到了对端端接, 做以下动作:
  - 等一定的时间 (时间值由实现决定);
  - 在全部准备激活的 TX Lane 上再发送一遍检测脉冲。
    - 如果检测结果和第一次一样, 做完以下动作后, 跳入下一状态 Discovery:
      - 如果 Lane0 上面没有检测到端接, 则根据检测结果进行 Lane reversal, 依据 TX Link 宽度, 将 Lane0 翻转到 Lane1/3/7。
      - 根据最终检测结果更新变量 *Tx\_M* 值留待后用。
    - 例如:
 

如果检测到了 Lane1 ~ Lane7 有端接, 则 Lane0 翻转到 Lane7, 更新 *Tx\_M* 值为 x4;

      - 否则, 下一状态是 Probe.Wait。
- 激活的 TX Lane 编号是 Lane *Tx\_0* 到 Lane *Tx\_M-1*。
- 依据当前 Lane0 的位置更新变量 *rx\_lane0\_id* 和 *tx\_lane0\_id*。

### 3.4.3.3 RXEQ\_Optimize

- 进入该状态后，链路两端退出电气空闲状态，开始发送 DLTB 和 AMCTL。
- 接收端检测 AMCTL 以进行 symbol lock；如果需要，RX 应该在所有准备激活的 RX Lane 上执行 RX 均衡参数适配。
- 在此状态中发送的 DLTB 除了 DLTB.Type = RXEQ\_Optimize 外，其它内容和 Discovery.Active 状态中发送的 DLTB 一致。
- 如果满足下面任一条件，则下一状态是 Discovery.Active：
  - 全部准备激活的 RX Lane 完成了 RX 均衡参数适配过程。
  - 在此状态超时。

超时值如下：

Data rate <= Data Rate 4:	48ms
Data Rate 4 < Data rate <= Data Rate 6:	72ms
Data Rate 6 < Data rate <= Data Rate 8:	120ms

### 3.4.3.4 Discovery

#### Discovery.Active

- 进入该状态后，链路两端退出电气空闲状态，开始发送 DLTB 和 AMCTL。  
在从电气空闲状态退出并发送 DLTB 之前，TX 必须要等它的共模电压稳定下来。
- 接收端检测 AMCTL 以进行 symbol lock。如果检测到 Lane 极性翻转了，则调整 Lane 的正负极性。
- TX 在所有激活的 TX Lane（如果跳过 Probe 阶段，则所有 TX Lane 为激活状态）上发送具有如下特征的 DLTB：
  - DLTB.Type = Discovery.Active。
  - Primary Port 将 DLTB.Link\_ID 设置为一个有效值；Secondary Port 设置 DLTB.Link\_ID = NULL。
  - DLTB.Lane\_ID 值设置为当前 Lane 的物理 Lane ID。
  - DLTB.TLW 设置为最新的 Tx\_M。
  - DLTB.RLW 设置为 Rx\_N。
  - DLTB.Data\_Rate\_Support\_1 和 DLTB.Data\_Rate\_Support\_2 广播包括本 Port 不准备使用的速率在内的所有支持的速率。
  - DLTB.FEC\_Mode\_Support symbol 广播本 Port 支持的所有 FEC 模式。
  - Primary Port 设置 DLTB.Port\_Type 为 1；Secondary Port 设置 DLTB.Port\_Type 为 0。
  - 如果端口类型协商没有被激活，则 DLTB.PortNego\_Random\_Value 设置为 NULL。
  - 当前 Link 的 Logic Lane0 的 DLTB.Lane\_ID 设为 0。

- 如果收到的 DLTB 中 Port\_Type 和发送的 Port\_Type 一样，则端口类型协商功能被激活。本 Port 产生一个 7 比特的随机数，并填入它发送的 DLTB.PortNego\_Random\_Value 域段。
- 激活的 RX Lane 编号是 Lane Rx\_0 到 Lane Rx\_N-1。
- 根据在任何激活的 RX Lanes 上收到的 DLTB 实时更新以下变量：
  - $Tx_M$  ( $Tx_M = \text{Min}\{\text{Rx.DLTB.RLW}, \text{Tx.DLTB.TLW}\}$ );
  - 根据在连续收到的 8 个 DLTB 中的 RX\_Lane0\_ID 值可能翻转 TX Lane0;
  - 根据当前 TX Lane0 位置更新变量  $tx\_lane0\_id$ 。
- 如果 Lane0 没有收到 8 个连续的，且满足以上全部条件的 DLTB，则根据激活的 RX lanes 来翻转 ( reverse ) RX Lane0 到 Lane7/3/1。
- 在全部激活的 RX Lane 上收到 8 个连续的 DLTB，并且在收到 1 个 DLTB 之后发送了至少一定数量的 DLTB ( 当从 Retrain 状态进入此状态时此值为 256 个 DLTB；如果是从别的状态进入的，此值为 1024 个 DLTB )，且收到的 DLTB 满足下列全部条件之后，下一个状态是 Discovery.Confirm：
  - DLTB.Type 是 Discovery.Active 或者 Discovery.Confirm。
  - 收到的 DLTB.Port\_Type 和发送的 DLTB.Port\_Type 不一样。
  - RX.DLTB.TLW 是标准的 Link Width 且小于或等于  $Rx_N$ 。
  - RX.DLTB.RLW 是标准的 Link Width 且小于或等于  $Tx_M$ 。
- 否则，在超时之后（如果是从 Retrain 或者 Config 状态进入本状态的，超时时间是 10us；如果从别的状态进入的，超时时间是 24ms），如果在任何激活的 RX Lane 上收到连续 8 个满足下列全部条件的 DLTB，则下一个状态是 Discovery.Confirm：
  - DLTB.Type 是 Discovery.Active 或者 Discovery.Confirm。
  - DLTB.TLW 和 DLTB.RLW 是有效的。
  - 在收到 1 个 DLTB 之后发送了至少 1024 个 DLTB。
  - 收到的 DLTB 中的 Port\_Type 和发送的不一样。
- 当状态机跳转时，需要完成以下动作：
  - 根据当前 RX Lane0 更新变量  $rx\_lane0\_id$
  - 更新变量  $Rx_N$  留待后面使用， $Rx_N = \text{Min}\{\text{RX.DLTB.TLW}, \text{TX.DLTB.RLW}\}$ 。
- 如果发送的 DLTB.Port\_Type 是 1，则本 Port 是 Primary Port。
- 激活的 RX Lane 编号是 Lane Rx\_0 到 Lane Rx\_N-1。
- 否则下一个状态是 Link\_Idle。

### Discovery.Confirm

- TX 在全部激活的 TX Lane 上发送带有有效 Link ID 的 DLTB：
  - DLTB.Type = Discovery.Confirm。
  - 当 DLTB.Port\_Type 是 Primary Port 时设置 DLTB.Link\_ID 为一个有效值；当 DLTB.Port\_Type 是 Secondary Port 时设置 DLTB.Link\_ID 为它从激活的 RX Lane0 上收到的 Link\_ID 值。

- DLTB.TLW 设置为变量  $Tx\_M$ 。
- DLTB.RLW 设置为变量  $Rx\_N$ 。
- DLTB.Data\_Rate\_Support\_1 和 DLTB.Data\_Rate\_Support\_2 保持和它在 Discovery.Active 状态中所发送的值相同。
- DLTB.FEC\_Mode\_Support 保持和它在 Discovery.Active 状态中所发送的值相同。
- Primary Port 设置 DLTB.Port\_Type 为 1; Secondary Port 设置 DLTB.Port\_Type 为 0。
- DLTB.PortNego\_Random\_Value 保持和在 Discovery.Active 状态发送的值一样。
- 如果任何激活的 RX Lane 收到 8 个连续的满足以上条件的 DLTB,但是 Rx.DLTB.RLW !=  $Tx\_M$ :
  - Port 可能根据收到的 8 个连续 DLTB 中的 RX\_Lane0\_ID 来翻转 TX Lane0。
  - 更新变量  $Tx\_M$  ( $Tx\_M = \min\{RX.DLTB.RLW, TX.DLTB.TLW\}$ )。
  - 根据当前 TX Lane0 位置更新变量  $tx\_lane0\_id$ 。
- 如果全部激活的 RX Lane 收到 8 个连续的 DLTB, 且在收到 1 个 DLTB 之后发送了至少 16 个 DLTB, 且收到的 8 个连续的 DLTB 满足下列全部条件, 则下一个状态是 Config:
  - DLTB.Type = Discovery.Confirm。
  - DLTB.Link\_ID 和发送的值相同。
  - RX.DLTB.TLW 是标准的 Link Width 且等于  $Rx\_N$ 。
  - RX.DLTB.RLW 是标准的 Link Width 且等于  $Tx\_M$ 。
  - 在端口类型协商激活的情况下 RX.DLTB.Port\_Type != TX.DLTB.Port\_Type。
- 如果  $Tx\_M \neq Rx\_N$ , 则变量  $symmetric\_link = 0$ ; 如果  $Tx\_M == Rx\_N$ , 则变量  $symmetric\_link = 1$ 。
- 否则, 在 48 ms 超时后下一个状态为 Link\_Idle。

### 3.4.3.5 Config

#### Config.Active

- TX 在所有于 Discovery 状态中确定激活的 Lane 上发送带有有效 Link ID 的 CLTB:
  - CLTB.Type = Config.Active。
  - CLTB.TLW 设置为变量  $Tx\_M$ 。
  - CLTB.RLW 设置为变量  $Rx\_N$ 。
  - 各个 Lane 的 Lane ID 设置为不重复的, 非 NULL 的值。在同一组激活的 Lane 中, 从 Lane  $Tx\_0$  到 Lane  $Tx\_M-1$  从 0 到 M-1 的顺序分配 Lane ID。
  - 根据 PHY Link Control 2 寄存器的 EQ Mode Control 域段值来设置 CLTB.EQ\_Mode 值。
- 如果在全部激活的 Lane 上收到 2 个连续的带有 non-NUL Link ID 以及 non-NUL Lane ID, 且匹配当前 Port TX Lane 上发送的 non-NUL Link ID 和 non-NUL Lane ID 的 CLTB (或者如果支持 Lane reversal 时, 匹配 reversed Lane ID ), 且在收到 1 个 CLTB 之后发送了至少 16 个 CLTB, 且收到的连续 2 个 CLTB 满足下列全部条件, 则下一个状态是 Config.Check:
  - CLTB.TLW 设置为变量  $Tx\_M$ 。
  - CLTB.RLW 设置为变量  $Rx\_N$ 。
  - 各个 Lane 的 Lane ID 设置为不重复的, 非 NULL 的值。在同一组激活的 Lane 中, 从 Lane  $Tx\_0$  到 Lane  $Tx\_M-1$  从 0 到 M-1 的顺序分配 Lane ID。
  - 根据 PHY Link Control 2 寄存器的 EQ Mode Control 域段值来设置 CLTB.EQ\_Mode 值。

- CLTB.Type 是 Config.Active 或者 Config.Check。
- Link ID 等于发送的 CLTB.Link\_ID。
- 收到的 CLTB.TLW 等于变量 Rx\_N。
- 收到的 CLTB.RLW 等于变量 Tx\_M。
- Lane ID 是 non-NUL 值。
- 否则，满足下列任一条件时，下一个状态是 Link\_Idle：
  - 2ms 超时。
  - 没有 Link 可以被配置。
  - 在全部激活的 Lane 上收到 2 个连续的 Discovery.Active 类型的 DLTB。

Primary Port 根据己方支持的 FEC 模式以及 Secondary Port 广播支持的 FEC 模式选择一种当前速率下双方都支持的 FEC 模式。

EQ mode 的确定由收到的两个连续的 CLTB 中的 EQ\_Mode 值以及自身发送的 CLTB 中的 EQ\_Mode 值来决定，如下表：

表 3-21 EQ 模式协商表

TX CLTB.EQ_Mode	RX CLTB.EQ_Mode	EQ mode
2'b00	2'b00	2'b00
2'b00	2'b01	2'b00
2'b00	2'b10	2'b00
2'b01	2'b00	2'b00
2'b01	2'b01	2'b01
2'b01	2'b10	2'b01
2'b10	2'b00	2'b00
2'b10	2'b01	2'b01
2'b10	2'b10	2'b10
2'b11	2'bxx	2'b00
2'bxx	2'b11	2'b00

### Config.Check

- TX 在所有于 Discovery 状态中确定激活的 Lane 上发送带有有效 Link ID 的 CLTB：
  - CLTB.Type = Config.Check。
  - CLTB.TLW 设置为变量 Tx\_M。
  - CLTB.RLW 设置为变量 Rx\_N。
  - 各个 Lane 的 Lane ID 设置为不重复的，非 NULL 的值。在同一组激活的 Lane 中，从 Lane Tx\_0 到 Lane Tx\_M-1 从 0 到 M-1 的顺序分配 Lane ID。

- 如果在全部激活的 Lane 上收到 2 个连续的带有 non-NUL Link ID 以及 non-NUL Lane ID，且匹配当前 Port 的 non-NUL Link ID 和 non-NUL Lane ID 的 CLTB，且在收到 1 个 CLTB 之后发送了至少 16 个 CLTB，且收到的 CLTB 满足下列全部条件，则下一个状态是 Config.Confirm：
  - CLTB.Type 是 Config.Check 或者 Config.Confirm。
  - Link ID 等于发送的 CLTB.Link\_ID。
  - 收到的 CLTB.TLW 等于变量 Rx\_N。
  - 收到的 CLTB.RLW 等于变量 Tx\_M。
- 否则，满足下列任一条件时，下一个状态是 Link\_Idle：
  - 2ms 超时。
  - 没有 Link 可以被配置。
  - 在全部激活的 Lane 上收到 2 个连续的 Discovery.Active 类型的 DLTB。

### Config.Confirm

- TX 在所有于 Discovery 状态中确定激活的 Lane 上发送带有有效 Link ID 的 CLTB：
  - CLTB.Type = Config.Confirm。
  - CLTB.TLW 设置为变量 Tx\_M。
  - CLTB.RLW 设置为变量 Rx\_N。
  - Lane ID 保持和 Config.Check 状态中发送的相同。
- 如果在全部激活的 Lane 上收到 2 个连续的带有有效 Link ID 以及 non-NUL Lane ID，且匹配当前 Port 的 non-NUL Link ID 和 non-NUL Lane ID 的 CLTB，且在收到 1 个 CLTB 之后发送了至少 16 个 CLTB，且收到的 CLTB 满足下列全部条件，则下一个状态是 Send\_NullBlock：
  - CLTB.Type == Config.Confirm。
  - Link ID 等于发送的 CLTB.Link\_ID。
  - 收到的 CLTB.TLW 等于变量 Rx\_N。
  - 收到的 CLTB.RLW 等于变量 Tx\_M。
- 配置好的 RX Lane number 是：Lane Rx\_0 到 Lane Rx\_N-1
- 配置好的 TX Lane number 是：Lane Tx\_0 到 Lane Tx\_M-1
- LMSM 跳变必须发生在准备发送 AMCTL 的边界。
- 否则，满足下列任一条件时，下一个状态是 Link\_Idle：
  - 2ms 超时。
  - 没有 Link 可以被配置。
  - 在全部激活的 Lane 上收到 2 个连续的 Discovery.Active 类型的 DLTB。

### 3.4.3.6 Send\_NullBlock

- 满足下列任一条件，则下一个状态是 Discovery：
  - 如果上层指示进入 Discovery。
    - 注 1：“上层指示”表示上层可选的控制 Port 重新配置 Link Width。
    - 注 2：LMSM 能通过此方式启动 Link Width 的增减，但是这种方式会中断正常的业务数据流。
  - 如果在任何配置的 Lane 上收到 2 个连续的 DLTB。
- 否则：
  - TX 在全部被配置好的 Lane 上发送 1 个 AMCTL with SDF 来启动 Data Stream，在此 AMCTL 之后发送 Null Block。
 

如果使能了 FEC mode，则全部的发送 Data Stream 需要进行 FEC 编码，FEC mode 在 Config 状态被确定。
  - RX 等待接收 Null Block。
  - LinkUp = 1b。
  - 如果在全部配置好的 Lane 上收到 8 个连续的 Null Block，且在收到 1 个 Null Block 之后发送了至少 16 个 Null Block。且满足下列全部条件，则下一个状态是 Link\_Active：
    - Null Block 必须是在 AMCTL with SDF 之后收到的。
    - 在开始 Data Stream 处理之前必须完成了 Lane 之间的 Deskew。
- 否则，在 2ms 超时之后：
  - 如果变量 Retrain\_num 小于 FFh，下一个状态是 Retrain.Active。
 

每跳入一次 Retrain，Retrain\_num 就加 1。
  - 否则，下一个状态是 Link\_Idle。

### 3.4.3.7 Link\_Active

这是可以发送数据链路层包的正常操作状态。

- 变量 LinkUp = 1b。
- 如果 Port 想启动速率的增加或者降低，设置变量 change\_speed = 1b。
- 如果在任何被配置的 Lane 上收到一个 AMCTL with EDF，则下一个状态是 Retrain.Active。
 

如果在任何被配置的 Lane 上收到一个 AMCTL with EDF，接收器数据路径应该停止处理数据。
- 如果在此状态时，当前操作的速率是端口两边在 Retrain 状态中广播的共同最高速率（如果进入 Link\_Active 之前没有进入过 Retrain 状态，则是 Discovery 状态中广播的共同最高速率，即 Data Rate 0），则变量 LinkReady = 1。
- 如果满足下列任一条件，在发送一个 AMCTL with EDF 之后，下一个状态是 Retrain.Active。
  - 如果 PHY Link Control 1 寄存器中的 Retrain Link 比特被设置为 1。
  - Port 想进入 Discovery 进行 Link Width 的增减，具体由特定的实现确定。

- 变量 `change_speed = 1` (通过 Retrain 进入 Change\_Speed 进行切速)。
- 在任何 Lane 上没有收到 AMCTL with EEI, 检测到或者推断出电气空闲, Port 可能跳入 Retrain.Active 状态。
- 如果 Port 检测到了 AMCTL 错误或者其它类型的 Framing error, Port 可能跳入 Retrain.Active 状态。

### 3.4.3.8 Retrain

#### Retrain.Active

- 变量 `LinkReady = 0`。
- 如果 Port 想做或者重做伴随着速率切换的 equalization, 且变量 `start_eq_init == 0`, 则 TX 在所有配置的 Lane 上发送具有如下特征的 RLTB:
  - RLTB.Type = Retrain.Active。
  - RLTB 中的均衡系数组值设置到当前链路速率相应的值。
  - 如果 Port 想重做 equalization, 则 RLTB.Request\_Equalization = 1。
  - RLTB.Remote\_Initial\_TX\_Preset 被设置为下一个速率的 DATA\_RATEx Control 寄存器中相应 Lane 的 Remote\_TX\_Preset\_Lanex 域段值。
  - 如果寄存器 DATA\_RATEx Control 中相应 Lane 的 Remote\_TX\_Preset\_Lanex 的值是 Reserved 或者不支持的值, TX 使用实现特定的方式选择一个支持的 Preset 值。
  - Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
  - Primary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播目标速率及目标速率以下的全部速率。(比如 Primary Port 和 Secondary Port 的共同最高速率为 106.25 Gbps, 工作在 PHY Mode-2, 使用 Full\_EQ 模式, 则 Primary Port 在此处的目标速率将依次分别为 25.78125 Gbps, 53.125 Gbps, 106.25 Gbps)。
  - 如果 Port 想要改变链路速率, 或者在任何配置的 Lane 上收到 8 个连续的 RLTB.Change\_Speed == 1 的 RLTB, 则变量 `change_speed = 1`。
  - 如果变量 `change_speed == 1`, 则 RLTB.Change\_Speed = 1。
  - RLTB.FEC\_Mode\_Ctrl 设置为下一个速率要使用的 FEC 模式。
- 如果 Port 只想做速率切换, 但不想做或者重做 equalization, 且变量 `start_eq_init == 0`, 则 TX 在所有配置的 Lane 上发送具有如下特征的 RLTB:
  - RLTB.Type = Retrain.Active。
  - RLTB 中的均衡系数组值设置到当前链路速率相应的值。
  - RLTB.Request\_Equalization = 0。
  - RLTB.Remote\_Initial\_TX\_Preset = 0。

- Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
- Primary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播目标速率及目标速率以下的全部速率。(比如 Primary Port 和 Secondary Port 的共同最高速率为 106.25 Gbps, 工作在 PHY Mode-2, 想要切速到 53.125 Gbps, 则 Primary Port 在此处的目标速率是 53.125 Gbps 及以下速率, 不广播 106.25 Gbps)。
- 如果 Port 想要改变链路速率, 或者在任何配置的 Lane 上收到 8 个连续的 RLTB.Change\_Speed == 1 的 RLTB, 则变量 *change\_speed* = 1。
- 如果变量 *change\_speed* == 1, 则发送的 RLTB.Change\_Speed = 1。
- RLTB.FEC\_Mode\_Ctrl 设置为下一个速率要使用的 FEC 模式。
- 如果 Port 想做速率切换及做 equalization, 且变量 *start\_eq\_init* == 1, 则 TX 在所有配置的 Lane 上发送具有如下特征的 RLTB:
  - RLTB.Type = Retrain.Active。
  - RLTB 中的均衡系数值设置到当前链路速率相应的值。
  - RLTB.Request\_Equalization = 0。
  - RLTB.Remote\_Initial\_TX\_Preset = 0。
  - Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
  - Primary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播当前速率及当前速率以下的全部速率。
  - 变量 *change\_speed* = 0。
  - RLTB.FEC\_Mode\_Ctrl 设置为当前 Port 使用的 FEC 模式。
- 如果想要做或者重做 equalization, 对 Primary Port 来说下一个状态是 EQ.Coarsetune\_Confirm, 对 Secondary Port 来说下一个状态是 EQ.Coarsetune\_Active:
  - Port 在跳进 Equalization 状态之前发送不超过 2 个 RLTB。
  - 操作在某个速率的 Port 使用本速率的 TX Preset 值发送数据, 每条 Lane 的本速率 TX Preset 值由如下规则确定:
    - 如果在最近跳转的 Retain.Confirm 状态中收到 8 个连续的包含有效 TX Preset 值的 Retain.EQ\_Initial 类型的 RLTB, 则此 TX Preset 值被使用。
    - 如果寄存器 DATA\_RATEx Control 中相应 Lane 的 Local\_TX\_Preset\_Lanex 域段包含的 TX Preset 值有效的话, 则使用这个 TX Preset 值。
    - 否则, 使用实现特定的方法获得有效的 TX Preset 值。
- 否则:
  - TX 在所有配置的 Lane 上发送具有如下特征的 RLTB:
    - RLTB.Type = Retain.Active。

- RLTB 中的均衡系数值设置到当前链路速率相应的值。
- RLTB.Request\_Equalization = 0。
- RLTB.Remote\_Initial\_TX\_Preset = 0。
- Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
- Primary Port 通过 RLTB.Data\_Rate\_Support\_1 及 RLTB.Data\_Rate\_Support\_2 广播当前速率及当前速率以下的全部速率。
- RLTB.FEC\_Mode\_Ctrl 设置为 Port 在当前速率想使用的 FEC 模式。
- 变量 *change\_speed* = 0。
- TX 使用最近一次 equalization 过程中协商好的 EQ coefficient 设置。
- 如果是从 Equalization 状态进入本状态的，且在 EQ.Passive 状态中接受的是 Preset 请求，那么发送的 RLTB 中设置 RLTB.Local\_TX\_Preset 为最新一次在 EQ.Passive 状态中接受的 Preset 值。
- 如果在全部配置的 Lane 上收到 8 个连续的 RLTB，RLTB.Change\_Speed 值等于变量 *change\_speed* 值；且在收到一个 RLTB 之后发出了至少 16 个 RLTB 之后，则下一个状态是 Retrain.Confirm。
- 如果是从 Equalization 状态进入本状态的，本 Port 评估所有 Lane 上收到的 RLTB 中的均衡 coefficients 或者 preset，判断它们是否和在 Equalization 状态中协商好的最终参数相同；如果不同，则在 Retrain.Confirm 状态中发送的 RLTB.Request\_Equalization = 1。  
TX 使用最近一次均衡过程中协商好的均衡 coefficients 设置进行发送。
- 如果满足下列条件，则下一个状态是 Discovery：
  - 在任何配置的 Lane 上收到至少 2 个连续的 DLTB。
- 允许 Primary Port 在此状态评估 Pre-FEC BER 值以及重新协商 FEC 模式，通过测量链路的 Pre-FEC BER，Port 可以选择匹配当前 Pre-FEC BER 的 FEC 模式。Primary Port 允许通过统计 LTB CRC 错误来测量 Pre-FEC BER。Port 测量 Pre-FEC BER 来选择 FEC 模式时满足下列要求：
  - 如果想测量 Pre-FEC BER，Port 设置 RLTB.Pre-FEC\_BER\_Measurement\_Status = 2'b01。
  - Port 在完成 Pre-FEC BER 测量及 FEC 模式选择之后，设置 RLTB.Pre-FEC\_BER\_Measurement\_Status = 2'b10。

注：Port 必须在 22ms 内完成 Pre-FEC BER 测量及 FEC 模式选择。

  - 在完成最终 FEC 模式选择后，Port 在 RLTB 中广播所选择的 FEC 模式。
  - 如果不想测量 Pre-FEC BER，而想使用软件来配置 FEC 模式，Port 设置 RLTB.Pre-FEC\_BER\_Measurement\_Status = 2'b00。

- 否则，在 24ms 超时之后：
  - 如果在任何配置的 Lane 上收到 8 个连续的 RLTB.Change\_Speed == 1 的 RLTB，且当前速率高于 Data Rate 0 或者在发送和接收的 RLTB 中广播的共同速率比当前速率更高，则下一个状态是 Retrain.Confirm。
  - 如果在进入 Retain 后 Link 当前的速率没有切换到目标速率，且当前操作的速率高于 Data Rate 0，则下一个状态是 Change\_Speed。在离开 Change\_Speed 之后新速率变为 Data Rate 0。
  - 如果在进入 Retain 后 Link 当前的速率切换到了目标速率，且当前操作的速率高于 Data Rate 0，则下一个状态是 Change\_Speed。在离开 Change\_Speed 之后新速率变回到进入 Retain 状态时的速率。
  - 当变量 *fix\_data\_rate\_mode* == 1 时，如果自进入此子状态 ( Retain.Active ) 以来任何配置的 Lane 收到至少两个连续的 RLTB 或者 DLTB，则下一状态是 Discovery。
  - 当变量 *fix\_data\_rate\_mode* == 0 时，如果满足下列条件，则下一状态是 Discovery：
    - 自进入此子状态( Retain.Active )以来在任何配置的 Lane 收到至少两个连续的 RLTB 或者 DLTB；
    - 当前速率是 Data Rate 0；
    - 变量 *changed\_speed\_retrain* == 0，且收到的 RLTB.Change\_Speed == 0。
  - 否则，下一个状态是 Link\_Idle。

### Retain.Confirm

- 如果 Port 想做或者重做伴随着速率切换的 equalization，且变量 *start\_eq\_init* == 0，则 TX 在所有配置的 Lane 上发送具有如下特征的 RLTB：
  - RLTB.Type = Retain.EQ\_Initial。
  - 将当前 Lane 上 RLTB.Local\_HS 和 RLTB.Local\_LS 设置为下一个速率要用的值。
  - 如果 Port 想重做 equalization，则 RLTB.Request\_Equalization 设置为 1。
  - RLTB.Remote\_Initial\_TX\_Preset 被设置为下一个速率的 DATA\_RATEEx Control 寄存器中相应 Lane 的 Remote\_TX\_Preset\_Lanex 值。
  - 如果寄存器 DATA\_RATEEx Control 中相应 Lane 的 Remote\_TX\_Preset\_Lanex 的值是 Reserved 或者不支持的值，TX 使用实现特定的方式选择一个支持的 Preset。
  - Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
  - Primary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播目标速率及目标速率以下的全部速率。
  - 如果 Port 想要改变链路速率，或者在任何配置的 Lane 上收到 8 个连续的 RLTB.Change\_Speed == 1 的 RLTB，则变量 *change\_speed* = 1。
  - 如果变量 *change\_speed* == 1，则发送的 RLTB.Change\_Speed = 1。

- RLTB.FEC\_Mode\_Ctrl 设置为下一个速率要使用的 FEC 模式。
- 否则，TX 在所有配置的 Lane 上发送具有如下特征的 RLTB：
  - RLTB.Type = Retrain.Confirm。
  - 如果 Port 想重做 equalization，则 RLTB.Request\_Equalization 设置为 1。
  - Secondary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播想要使用的全部速率。
  - Primary Port 通过 RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2 广播目标速率及目标速率以下的全部速率。
  - 如果变量 *change\_speed == 1*，则发送的 RLTB.Change\_Speed = 1。
- 如果满足以下全部条件，TX 在全部配置的 Lane 上发送类型为 Retrain.EQ\_Initial 的 RLTB：
  - Primary Port 和 Secondary Port 在 Retrain 及 Discovery 状态广播了 Data RateX ( X>0 ) 速率，且在进入此状态之前在任何配置的 Lane 上收到了连续 8 个 RLTB.Change\_Speed == 1 的 RLTB。
  - 变量 *eq\_complete\_data\_rate(x>0)* 为 0，或者寄存器 PHY Link Control 2 的 Perform Equalization 域段为 1，或者实现特定的机制决定要执行 equalization。
  - 当前操作的速率低于 Data RateX ( X>0 )。
- 进入此子状态后，变量 *start\_eq\_init = 0*。
- 如果满足下列所有条件，下一个状态是 Change\_Speed：
  - 在任何配置的 Lane 上收到 8 个连续的 RLTB.Change\_Speed == 1，类型为 Retrain.Confirm 或 Retrain.EQ\_Initial 的 RLTB。
  - 在收到一个 RLTB.Change\_Speed == 1 的 RLTB 之后，在同一个配置的 Lane 上发出了 128 个 RLTB.Change\_Speed = 1 的 RLTB。
  - 当前速率高于 Data Rate 0，或者发送的 RLTB 及至少 8 个连续收到的 RLTB 中广播的最高速率都高于 Data Rate 0。

在 Change\_Speed 中改变到的新速率是 Link 两端的 Port 支持的最高共同速率。

如果速率是 Data Rate 1 或者更高，且变量 *start\_eq\_init == 1*：

  - 在要执行 equalization 的速率下开始发送数据时，Secondary Port 使用在 Retrain.Confirm 状态期间收到的 8 个连续的 RLTB 中的 TX preset 值作为它的 TX setting，并确保满足相应的 preset 参数定义。
  - 在要执行 equalization 的速率下开始发送数据时，收到 Reserved 或者不支持的 preset 值的 Lane 使用实现特定的方式选择一个合法的 TX preset 设置它的 TX 来发送 pattern。
- 如果满足下列任一条件，则下一个状态是 Discovery：
  - 如果任何配置的 Lane 上收到 8 个连续的 DLTB。
  - 如果上层指示进入。

“上层指示进入”在这里的意思是上层设置寄存器 PHY Link Control 1 中的 Retrain Link 比特来重新配置 Link Width。

在进入 Discovery 状态时变量 *changed\_speed\_retrain* = 0。

- 如果满足下列所有条件，则下一个状态是 Send\_NullBlock：
  - 全部配置的 Lane 上收到 8 个连续的类型为 Retain.Confirm 的 RLTB，在每一条 Lane 上的速度支持标识符 (RLTB.Data\_Rate\_Support\_1 和 RLTB.Data\_Rate\_Support\_2) 都相等，且满足下列条件之一：
    - 在收到的 8 个连续的 RLTB 中 RLTB.Change\_Speed == 0。
    - 当前速率是 Data Rate 0；且收到的 RLTB 或者发送的 RLTB 中没有广播高于 Data Rate 0 的速率。
  - 在收到一个 RLTB 之后，发送出去了 16 个没有被 AMCTL 打断的 RLTB。
  - 当前处于预备发送 AMCTL with SDF 的边界。
 在进入 Send\_NullBlock 时变量 *changed\_speed\_retrain* 和 *change\_speed* 复位为 0。
- Port 需要判断对端在前一个状态是否做了 Pre-FEC BER 测量以及 FEC 模式协商。  
如果做了，且接收到的 FEC 模式收益低于所要求的 FEC 模式（即 FEC 模式纠错能力不够），则 Port 需要选择使用默认的 FEC 模式并在发送的 RLTB.FEC\_Mode\_Ctrl 域段反映默认的 FEC 模式且将 RLTB.FEC\_Mode\_Reject 置为 1。  
否则使用对端指定的 FEC 模式，并在发送的 RLTB.FEC\_Mode\_Ctrl 中反映此 FEC 模式且将 RLTB.FEC\_Mode\_Reject 置为 0。  
如果对端收到的 RLTB.FEC\_Mode\_Reject == 1，则对端使用默认的 FEC 模式。默认的 FEC 模式是 RS(128,120,T=4)。
- 如果从 Link\_Active 进入 Retain 以来速率已经改变到了共同的协商速率，且在任何配置的 Lane 上收到一个 AMCTL with EEI 或者检测到/推断出电气空闲，且自进入 Retain.Confirm 以来没有配置的 Lane 收到过类型为 Retain.Confirm 的 RLTB，则下一个状态是 Change\_Speed。  
在离开 Change\_Speed 之后，新速率将变回到从 Link\_Active 进入 Retain 时的速率。
- 如果从 Link\_Active 进入 Retain 以来速率没有改变到共同的协商速率，且在任何配置的 Lane 上收到一个 AMCTL with EEI 或者检测到/推断出电气空闲，且自进入 Retain.Confirm 以来没有配置的 Lane 收到过类型为 Retain.Confirm 或者 Retain.EQ\_Initial 的 RLTB，则下一个状态是 Change\_Speed。  
在离开 Change\_Speed 之后，新速率将切换到 Data Rate 0。
- 否则，在 48ms 超时之后：
  - 如果变量 *fix\_data\_rate\_mode* == 1，如果自进入此子状态 (Retain.Confirm) 以来在任何配置的 Lane 上收到至少 2 个连续的 RLTB 或者 DLTB，下一个状态是 Discovery。
  - 如果变量 *fix\_data\_rate\_mode* == 0，如果满足以下全部条件，则下一个状态是 Discovery：

- 自进入此子状态( Retrain.Confirm )以来在任何配置的 Lane 上收到至少 2 个连续的 RLTB 或者 DLTB。
- 当前速率是 Data Rate 0。
- 变量 *changed\_speed\_retrain* == 0，且收到的 RLTB.Change\_Speed == 0。
- 如果变量 *Retain\_num* 小于 0xFF 且当前速率大于 Data Rate 0，则下一个状态是 Send\_NullBlock。  
进入 Send\_NullBlock 状态后，变量 *changed\_speed\_retrain* = 0。
- 否则下一个状态是 Link\_Idle。

### 3.4.3.9 Change\_Speed

光链路模式不支持此状态。

- 在所有激活的 Lanes 上发送一个 AMCTL with EEI 之后，TX 进入电气空闲状态，直到本端 RX 也进入了电气空闲状态：
  - 在切速成功的情况下，TX 保持在电气空闲状态一定的时间（时间值由实现决定）。
  - 在切速不成功的情况下，TX 保持在电气空闲状态一定的时间（时间值由实现决定，但需要大于切速成功情况下的时间值）。
- 只有在 RX 进入 Electrical Idle 之后，且 TX 满足以上条件不需要呆在 Electrical Idle 之后，才允许切换到新速率。  
注：如果 Link 已经运行在所支持的最高速率，此时 Change\_Speed 状态被执行，但是速率不改变。
- 在成功切速后，下一个状态是 Retrain.Active。新速率根据以下规则确定：
  - 如果是从 Retain.Confirm 进入此状态的，且随后切速成功，则新速率切换到所有配置的 Lane 上广播的两端公共最高速率，变量 *changed\_speed\_retrain* = 1。
  - 另外如果是自 Link\_Active 状态进入 Retain 以来的第二次进入此状态，则新速率切换为从 Link\_Active 状态进入 Retain 时的速率，变量 *changed\_speed\_retrain* = 0。
  - 否则新速率将切换为 Data Rate 0，变量 *changed\_speed\_retrain* = 0。
- 变量 *change\_speed\_retrain* = 0。切速后的新速率被反映在 PHY Link Status 1 寄存器的 Current Link Speed 域段。
- 否则，在 48ms 超时后，下一个状态是 Link\_Idle。

### 3.4.3.10 Equalization

#### EQ.CoarseTune\_Active

进入此子状态时，当前速率下的寄存器 State 1 中如下域段需要置为 0：Equalization CoarseTune Phase Complete, Equalization Active Phase Successful, Equalization Passive Phase Successful, Equalization Complete。

如下变量置为 0:  $eq\_complete\_data\_rate1 \sim eq\_complete\_data\_ratex$  ( $eq\_complete\_data\_ratex$  由本 Port 支持的最高速率决定),  $start\_eq\_init$ 。

注: 如果 RX Lane 的顺序和 TX Lane 的顺序不同, 则应在均衡过程中翻转均衡信息。

- TX 在所有配置好的 Lane 上发送具有以下特征的 ELTB:
  - ELTB.Type = Equalization。
  - ELTB.Current\_EQ\_Phase = Coarsetune\_Active。
  - 每条 Lane 的 TX\_Preset 域段设置为它当前相应的 TX Preset 设置值。
  - 每条 Lane 的 Pre-Cursor 和 Post-Cursor 域段设置为它当前速率相应的 TX Pre-Cursor 和 Post-Cursor 值。
- 进入本状态后, 端口允许等一定的时间 (时间值由实现决定) 开始评估接收到的信息。如果 RX 完成了 AMCTL 锁定, 且全部配置的 Lane 收到 2 个连续的满足以下所有条件的 ELTB, 则下一个状态是 EQ.Coarsetune\_Confirm:
  - ELTB.Type == Equalization。
  - ELTB.Current\_EQ\_Phase == Coarsetune\_Confirm。
  - 如果需要, RX 完成 Rx Equalization 自适应过程。
- 否则在 12ms 超时之后:
  - 如果寄存器 PHY Link Control 11 中域段 Skip EQ Coarsetune Enable 所指示的当前速率对应比特是 1 时, 下一个状态是 EQ.Coarsetune\_Confirm。
  - 否则, 下一个状态是 Change\_Speed:
    - 变量  $successful\_speed\_change = 0$ ,
    - 对应速率的 State 1 寄存器的 Equalization Complete = 1, Equalization Coarsetune Phase Complete = 1。

#### **EQ.Coarsetune\_Confirm**

进入此子状态时, 当前速率下的寄存器 State 1 如下域段需要置为 0: Equalization Coarsetune Phase Complete, Equalization Active Phase Successful, Equalization Passive Phase Successful, Equalization Complete。

如下变量置为 0:  $eq\_complete\_data\_rate1 \sim eq\_complete\_data\_ratex$  ( $eq\_complete\_data\_ratex$  由本 Port 支持的最高速率决定),  $start\_eq\_init$ 。

- TX 在所有配置好的 Lane 上发送具有以下特征的 ELTB:
  - ELTB.Type = Equalization。
  - ELTB.Current\_EQ\_Phase = Coarsetune\_Confirm。
  - 每条 Lane 的 TX\_Preset 域段设置为它当前相应的 TX Preset 设置值。
  - 每条 Lane 的 Pre-Cursor 和 Post-Cursor 域段设置为它当前速率相应的 TX Pre-Cursor 和 Post-Cursor 值。

**对于 Primary Port:**

- 进入本状态后，端口允许等一定的时间（时间值由实现决定）开始评估接收到的信息。
- 如果 RX 完成了 AMCTL 锁定，且全部配置的 Lane 收到 2 个连续的满足以下所有条件的 ELTB，且本 Port 需要执行 EQ.Active 和 EQ.Passive 阶段，则下一个状态是 EQ.Passive：
  - ELTB.Type == Equalization。
  - ELTB.Current\_EQ\_Phase == Coarsetune\_Confirm。
  - 如果需要，RX 完成 Rx Equalization 自适应过程。
 在跳转到 EQ.Passive 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete，Equalization Coarsetune Phase Successful 比特置为 1。
- 否则，如果如果 RX 完成了 AMCTL 锁定，且全部配置的 Lane 收到 2 个连续的满足以下所有条件的 ELTB，且本 Port 不需要执行 EQ.Active 和 EQ.Passive 阶段，则下一个状态是 Retrain.Active：
  - ELTB.Type = Equalization。
  - ELTB.Current\_EQ\_Phase = Coarsetune\_Confirm。
  - 如果需要，RX 完成了 Rx Equalization 自适应过程。
 在跳转到 Retrain.Active 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete，Equalization Coarsetune Phase Successful，Equalization Active Phase Successful，Equalization Passive Phase Successful，Equalization Complete 比特置为 1。
- 否则在 24ms 超时之后：
  - 如果寄存器 PHY Link Control 11 中域段 Skip EQ Coarsetune Enable 所指示的当前速率对应比特是 1 时，下一个状态是 EQ.Passive。
 在跳转到 EQ.Passive 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete，Equalization Coarsetune Phase Successful 比特置为 1。
- 否则，下一个状态是 Change\_Speed：  
变量 *successful\_speed\_change* = 0；  
当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete，Equalization Complete 比特置为 1。

**对于 Secondary Port:**

- 如果全部配置的 Lane 收到 2 个连续的满足以下所有条件的 ELTB，则下一个状态是 EQ.Active：
  - ELTB.Type == Equalization。
  - ELTB.Current\_EQ\_Phase == Passive\_Phase。
 在跳转到 EQ.Active 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete，Equalization Coarsetune Phase Successful 比特置为 1。

- 否则，如果全部配置的 Lane 收到 8 个连续的 RLTB，则下一个状态是 Retrain.Active。  
在跳转到 Retrain.Active 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete , Equalization Coarsetune Phase Successful , Equalization Complete 比特置为 1。
- 否则在 12ms 超时之后：
  - 如果寄存器 PHY Link Control 11 中域段 Skip EQ Coarsetune Enable 所指示的当前速率对应比特是 1 时，且本 Port 想执行 EQ.Active 和 EQ.Passive 阶段，则下一个状态是 EQ.Active。  
在跳转到 EQ.Active 状态之前完成下列动作：当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete , Equalization Coarsetune Phase Successful 比特置为 1。
  - 否则，下一个状态是 Change\_Speed：  
变量 *successful\_speed\_change* = 0；  
当前速率下的寄存器 State 1 的 Equalization Coarsetune Phase Complete , Equalization Complete 比特置为 1。

在完成 EQ.Coarsetune\_Confirm 阶段之后，Primary Port 和 Secondary Port 的 BER 都应该小于  $10^{-4}$ 。

### EQ.Passive

- TX 在所有配置好的 Lane 上发送具有以下特征的 ELTB：
  - ELTB.Type = Equalization。
  - ELTB.Current\_EQ\_Phase 设置为 Passive\_Phase
- 如果收到 2 个连续的 ELTB.Current\_EQ\_Phase == Active\_Phase 的 ELTB：
  - 如果收到的 ELTB 中的 Preset 或者 coefficient 是合法的：
    - 设置 TX EQ 到所要求的 Preset 或者 coefficient，且确保新设定的 EQ 值在一定的时间（时间值由实现决定）之内在 TX 引脚上生效。
    - 在发送的 ELTB 中，如果 EQ 请求是 Preset 形式，则 ELTB.Preset\_Mode\_En = 1；如果是 coefficient 形式，则 ELTB.Preset\_Mode\_En = 0。无论是 Preset 还是 coefficient 请求，Pre-Cursor 和 Post-Cursor 域段都填充为和当前设定的 TX EQ 相应的值。  
ELTB.EQ\_Reject = 0。
  - 如果收到的 ELTB 中的 Preset 或者 coefficient 是非法的：
    - 不改变 TX EQ 设定。
    - 在发送的 ELTB 中，如果 EQ 请求是 Preset 形式，则 ELTB.Preset\_Mode\_En = 1；如果是 coefficient 形式，则 ELTB.Preset\_Mode\_En = 0；通过 Pre-Cursor 和 Post-Cursor 值反映所收到的 Preset 或者 coefficient 请求值，ELTB.EQ\_Reject = 1。
- 对于 Primary Port，如果在所有配置好的 Lane 上收到 2 个连续的 ELTB.Current\_EQ\_Phase == Passive\_Phase 的 ELTB，则下一个状态是 EQ.Active。
  - 当前速率下的寄存器 State 1 中 Equalization Passive Phase Successful 置为 1。

- 如果在所有配置好的 Lane 上收到 2 个连续的 RLTB，则下一个状态是 Retrain.Active。
  - 当前速率下的寄存器 State 1 中 Equalization Passive Phase Successful 置为 1。
  - 保持 TX EQ 设定为最近收到的合法的值。
- 在超时（速率<=Data Rate 4 时：32ms；速率> Data Rate 4 时：64ms）之后，下一个状态是 Change\_Speed：
  - 变量 *successful\_speed\_change* = 0。
  - 当前速率下的寄存器 State 1 中 Equalization Complete = 1。

**EQ.Active**

- TX 在所有配置好的 Lane 上发送具有以下特征的 ELTB：
  - ELTB.Type 是 Equalization。
  - ELTB.Current\_EQ\_Phase = Active\_Phase
- 如果是光链路模式，Port 不需要执行远端 FFE training，但是需要做 RX CTLE 适应调整。
- 如果不是光链路模式，Port 需要根据以下要求在每个配置的 Lane 上独立进行 EQ 评估：
  - 如果想使用 Preset 方式，则在发送的 ELTB 中将 ELTB.Preset\_Mode\_En 设置为 1，将 ELTB.Request\_TX\_Preset 设置为希望评估的 Preset 值。
  - 如果想使用 coefficient 方式，则在发送的 ELTB 中将 ELTB.Preset\_Mode\_En 设置为 0，将 Pre-Cursor 和 Post-Cursor 域段设置为希望评估的 coefficient 值。
  - 一旦更新了一个新 EQ 值评估请求，此请求必须保持一定的时间（时间值由实现决定），或者等到这个评估完成。
  - 等待一定的时间（时间值由实现决定），以确保对端 Port（处于 EQ.Passive 状态的 Port）在接受请求后已经使用所请求的 EQ setting 发送码流，此后本 Port 可以通过接收码流评估当前 setting。
  - 如果收到 2 个满足下列条件的连续的 ELTB：
    - ELTB.Current\_EQ\_Phase == Passive\_Phase。
    - ELTB.EQ\_Reject == 0。
    - ELTB.Request\_TX\_Preset（使用 Preset 方式时）或者 Pre-Cursor 和 Post-Cursor 域段值等于本端所请求的 EQ 值。

这意味着对端 Port 已经接受了所请求的 EQ 值，本端 RX 可以开始评估当前的 EQ setting。

当前的 EQ setting 可以被考虑作为候选的最终 EQ setting。如果当前 setting 没有被选为最终 EQ setting，本 Port 可以尝试评估下一组 EQ setting。

- 如果收到 2 个满足下列条件的连续的 ELTB：
  - ELTB.Current\_EQ\_Phase == Passive\_Phase。
  - ELTB.EQ\_Reject == 1。

- ELTB.Request\_TX\_Preset ( 使用 Preset 方式时 ) 或者 Pre-Cursor 和 Post-Cursor 域段值等于本端所请求的 EQ 值。

这意味着对端 Port 已经拒绝了所请求的 EQ 值，本端 RX 不评估当前的 EQ setting。

当前的 EQ setting 不能被考虑作为候选的最终 EQ setting，本 Port 可以尝试评估下一组 EQ Setting。

- 如果 RX 在 2ms 内没有收到 2 个连续的 ELTB.Current\_EQ\_Phase == Passive\_Phase 的 ELTB，且本状态还没有到 24ms 超时，本 Port 需要放弃当前 EQ setting，并开始发出下一个 EQ setting 请求。
- 本 Port 在 24ms 内完成所有的 EQ setting 评估。
- 对于 Primary Port，如果所有配置好的 Lane 达到了优化的 EQ setting，且其 RX 如有必要完成了 RX EQ 参数调优适配过程，下一个状态是 Retrain.Active。  
当前速率下的 State 1 寄存器的 Equalization Active Successful，Equalization Complete 比特设置为 1。
- 对于 Secondary Port，如果所有配置好的 Lane 达到了优化的 EQ setting，且其 RX 如有必要完成了 RX EQ 参数调优适配过程，下一个状态是 EQ.Passive。  
当前速率下的 State 1 寄存器的 Equalization Active Successful，Equalization Complete 比特设置为 1。
- 在超时（速率 $\leq$  Data Rate 4 时：24ms；速率 $>$  Data Rate 4 时：48ms。超时值的容差要求是 0ms 到 2ms 之间）之后，下一个状态是 Change\_Speed：
  - 变量 successful\_speed\_change = 0。
  - 当前速率下的 State 1 寄存器的 Equalization Complete 比特设置为 1。

# 4 数据链路层

## 4.1 概述



图 4-1 数据链路层概览

如上图所示，数据链路层位于物理层和网络层之间，为上层提供点到点数据包可靠传送的服务。数据链路层主要支持如下特性：

1. 数据链路层控制块（Data Link Layer Control Block, DLLCB）和数据链路层数据包（Data Link Layer Data Packet, DLLDP）的封装和解析能力，分别实现链路管理控制以及正常数据收发功能，相关内容见 4.3 节；
2. CRC（Cyclic Redundancy Check）和 Non-CRC 两种封装模式，应用于不同物理链路质量情况，可选通过 CRC 模式实现更高可靠性，或通过 Non-CRC 模式实现更高链路带宽利用率和更低时延，相关内容见 4.3 节；
3. 虚拟通道（Virtual Lane, VL）机制，数据链路层每个链路提供最多 16 个 VL，为上层提供流量隔离能力，相关内容见 4.5 节；
4. 信用流控能力，基于 VL 实现信用证流量控制，在发送端精确感知和使用接收端 Receive Buffer 空间，相关内容见 4.6 节；
5. 点到点重传能力，在接收端收到的 DLLCB/DLLDP 出错时发起重传请求，触发发送端重传，保证 DLLCB/DLLDP 的无损可靠传送，相关内容见 4.7 节。

## 4.2 数据链路状态机

数据链路状态机用于对链路状态进行管理和控制，其状态包括：DLL\_Disabled、DLL\_Param\_Init、DLL\_Credit\_Init 及 DLL\_Normal，状态机设计详见图 4-2。

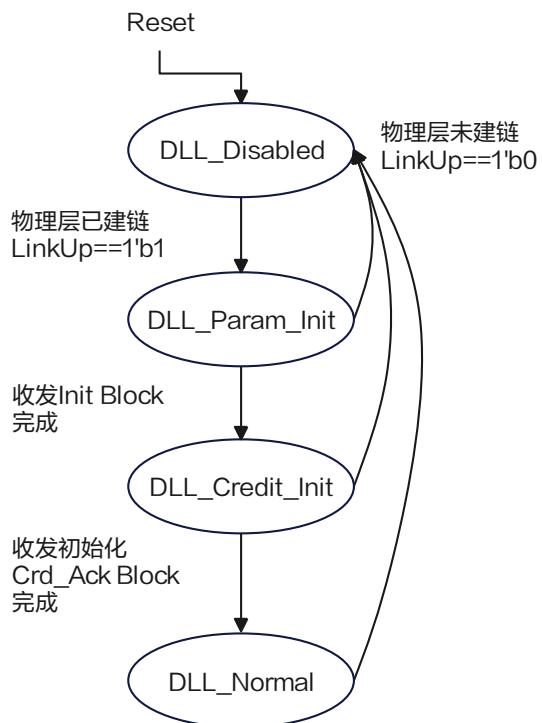


图 4-2 数据链路状态机

数据链路状态机中各状态定义如下：

1. **DLL\_Disabled**: 链路在 UB 设备级或 Port 级复位后的初始化状态。Entity 级复位不会导致链路状态机进入 **DLL\_Disabled** 状态，该状态下物理层未建链（物理层信号  $\text{LinkUp}==1'b0$ ），复位相关的要求见 10.6.1 节；
2. **DLL\_Param\_Init**: 参数初始化状态。物理层通告物理链路已建链（物理层信号  $\text{LinkUp}==1'b1$ ），数据链路层在此状态与对端进行信用证返还粒度、ACK 返还粒度和 VL 启用等相关参数的协商，相关内容见 4.4 节；
3. **DLL\_Credit\_Init**: 信用证初始化状态。成功完成参数协商后，数据链路层在此状态下进行信用证初始化，相关内容见 4.6.1 节；
4. **DLL\_Normal**: 数据通信状态。本端数据链路层能与对端数据链路层使用 DLLDP 进行通信。

本端数据链路层根据是否能与链路另一端传输 DLLDP，向网络层输出不同状态：

1. **DLL\_Status\_Down**: 本端数据链路层无法与链路另一端数据链路层传输 DLLDP；
2. **DLL\_Status\_Up**: 本端数据链路层可以与链路另一端数据链路层传输 DLLDP。

处于不同状态下的链路应符合如下要求：

1. **DLL\_Disabled**:
  - (1) 处于 **DLL\_Disabled** 状态时：
    - 向网络层报告 **DLL\_Status\_Down** 状态；
    - 丢弃来自本端网络层下发的 Packet 和本端物理层接收的 DLLCB/DLLDP；
    - 不生成 DLLCB。

(2) 跳转到 DLL\_Param\_Init 的条件为：物理层报告信号 LinkUp==1'b1。

## 2. DLL\_Param\_Init:

(1) 处于 DLL\_Param\_Init 状态时：

- 向网络层报告 DLL\_Status\_Down 状态；
- 丢弃来自本端网络层下发的 Packet 和本端物理层接收的 LLDP；
- 首先发送重传请求序列，直到接收到对端的重传应答序列后，再收发 Init Block 进行数据链路层初始化自协商。重传序列相关内容见 4.7.3 节，Init Block 相关内容见 4.3.3.9 节，自协商相关内容见 4.4 节。

(2) 跳转到 DLL\_Credit\_Init 状态的条件为：数据链路层完成初始化自协商并且物理层持续宣告信号 LinkUp==1'b1。

(3) 跳转到 DLL\_Disabled 状态的条件为：物理层报告信号 LinkUp==1'b0。

## 3. DLL\_Credit\_Init:

(1) 处于 DLL\_Credit\_Init 状态时：

- 向网络层报告 DLL\_Status\_Down 状态；
- 丢弃来自本端网络层下发的 Packet 和本端物理层接收的 LLDP；
- 收发 Crd\_Ack Block 进行数据链路层信用证初始化，信用证初始化相关内容见 4.6.1 节。

(2) 跳转到 DLL\_Normal 状态的条件为：数据链路层完成信用证初始化并且物理层持续宣告信号 LinkUp==1'b1。

(3) 跳转到 DLL\_Disabled 状态的条件为：物理层报告信号 LinkUp==1'b0。

## 4. DLL\_Normal:

(1) 处于 DLL\_Normal 状态时：

- 向网络层报告 DLL\_Status\_Up 状态；
- 发送和接收 DLLCB 和 LLDP。

(2) 跳转到 DLL\_Disabled 状态的条件为：物理层报告信号 LinkUp==1'b0。

## 4.3 DLLCB 和 LLDP 收发

### 4.3.1 DLLCB 和 LLDP 收发概述

数据链路层支持收发 DLLCB 和 LLDP，DLLCB 和 LLDP 均支持两种封装模式：Non-CRC 模式和 CRC 模式，两种模式应用于不同链路状况的场景。封装模式切换由物理层发起，数据链路层配合完成模式切换。BER 测量及封装模式切换相关内容见 3.4.2.8 节和 4.3.3.8 节。

### 4.3.2 DLLDP 格式

#### 4.3.2.1 DLLDP 的组成

链路两端的数据链路层以 DLLDP 为粒度将上层业务数据传输给对端, DLLDP 组成的最小单元为 Flit, DLLDP 最大长度为 512 Flits。当 DLLDP 长度小于等于 32 Flits 时, 使用 1 个对应 Flit 数量的数据链路层数据块 ( Data Link Layer Data Block, DLLDB ) 进行传输; 当 DLLDP 长度大于 32 Flits 时, 会将其切分为最多 16 个 DLLDB 分段传输。DLLDP 内第一个 DLLDB 称为 First Block, 最后一个 DLLDB 称为 Last Block, 其它 DLLDB 称为 Middle Block。每个 DLLDB 的最大长度为 32 Flits, DLLDP、DLLDB 和 Flit 对应关系见图 4-3。每个 Flit 大小固定为 20 Bytes, 不足 20 Bytes 时需要用 Padding 补齐, 发送端补齐 Padding 时每 bit 均填 0, 接收端忽略 Padding。

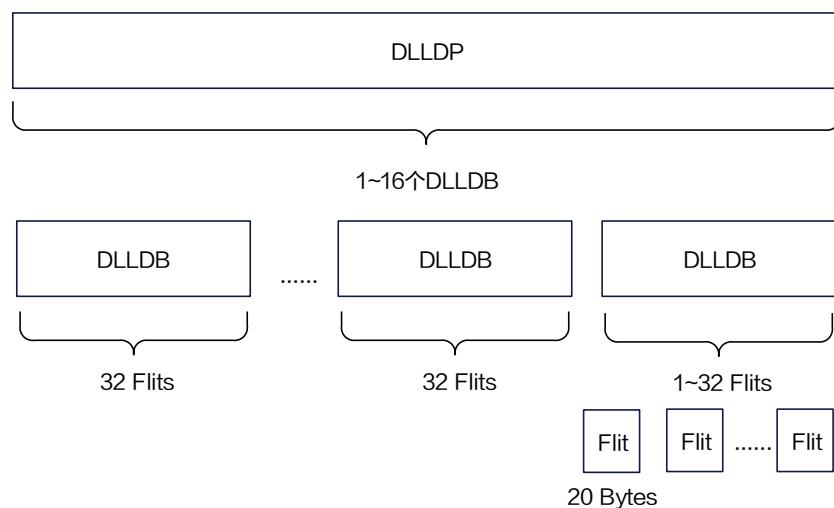


图 4-3 DLLDP、DLLDB 和 Flit 关系图

示例: CRC 模式 Padding 填充 Flit

如图 4-4, DLLDB 由 3 个 Flits 组成, 只需使用 Padding 填充满最后一个 Flit 即可, CRC 模式相关内容见 4.3.2.2 节。

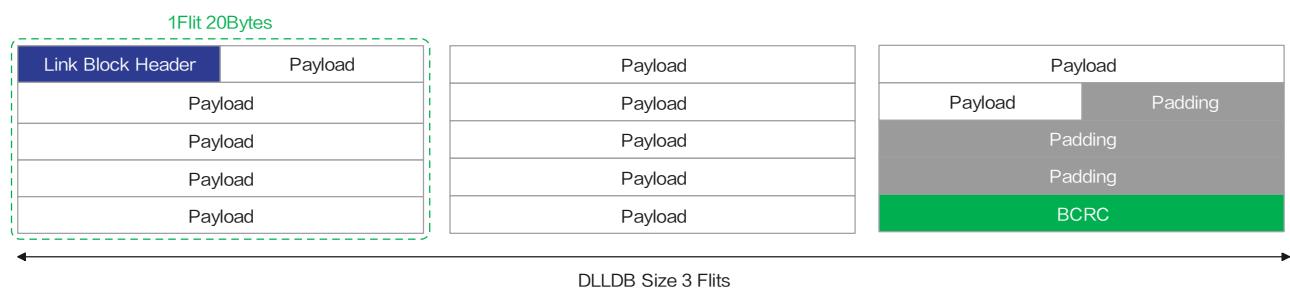


图 4-4 Padding 填充示例

### 4.3.2.2 CRC 模式

#### 4.3.2.2.1 CRC 模式 DLLDP 格式

CRC 模式下，DLLDP 中 First Block 的前 4 字节为 LPH ( Link Packet Header )，Middle Block 和 Last Block 的前 2 字节为 LBH ( Link Block Header )，每个 DLLDB 的最后一个 Flit 的后 4 字节为 BCRC ( Block Cyclic Redundancy Check )。CRC 模式可选启用物理层 FEC，当启用物理层 FEC 时，数据链路接收 Flit 后需要判断 FEC 解码是否成功，若 FEC 解码成功，则通过 LPH/LBH 完成定界、信用流控等功能，并通过 CRC 校验准确性；若 FEC 解码失败或 CRC 校验出错，则通过数据链路层重传机制进行重传。CRC 模式 DLLDP 格式示意图见图 4-5。



图 4-5 CRC 模式 DLLDP 格式

#### 4.3.2.2.2 LPH 格式

LPH 用于存放 DLLDP 的描述和控制信息，包括 DLLDP 的长度、是否返还信用证等信息。DLLDP 第一个 Flit 的格式见图 4-6，根据 Payload 长度不同，Payload[31:0]域段也可能为 BCRC。

Byte0								Byte1								Byte2								Byte3							
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
Link Packet Header																															
Payload[127:96]																															
Payload[95:64]																															
Payload[63:32]																															
Payload[31:0]/BCRC																															

图 4-6 DLLDP 第一个 Flit 格式

LPH 共占用 4 Bytes，其中各个域段的定义见表 4-1。

表 4-1 CRC 模式 LPH 域段

LPH			
字节编号	bit 编号	域段名	含义
0	7	CRD	指示 Block 是否返还信用证，1b'0 表示不返还，1b'1 表示返还，返还粒度在初始化过程中通过协商确定，参数初始化协商相关内容见 4.4 节。
	6	ACK	指示 Block 是否释放 Retry Buffer 空间，1b'0 表示不释放，1b'1 表示释放，释放粒度在初始化过程中通过协商确定，参数初始化协商相关内容 4.4 节。
	5	CRD_VL[3:0]	指示 CRD 域段返还的信用证所对应的 VL 编号。
	4		
	3		
	2		
	1	Reserved	保留字段，发送端默认填 0，接收端忽略。
	0	VL[3:0]	Block 的 VL 编号，VL 域段值由网络层携带到数据链路层，VL 相关内容见 4.5 节。
1	7		
	6		
	5		
	4	Reserved	保留字段，发送端默认填 0，接收端忽略。
	3	CFG[3:0]	数据链路层通过 CFG 域段区分 DLLCB 和 DLLDP。 CFG == 0 时为 DLLCB，由数据链路层生成并填充； CFG == 3/4/5/6/7/9 时为 DLLDP，CFG 域段值由网络层携带到数据链路层，具体类型由上层定义，链路层不感知。
	2		
	1		
	0		

LPH			
字节编号	bit 编号	域段名	含义
			CFG == 其它，保留。 注：DLLDP 中的 CFG 值必定不为 0。
2	7	RT	Routing Type，路由类型。RT 域段值由网络层携带到数据链路层，相关内容见 5.3.2 节。
	6		
	5	PLENGTH[13:10]	指示 DLLDB 数量：0~15 分别指示 1~16 个 DLLDB。
	4		
	3		
	2		
	1		指示最后一个 DLLDB 的 Flit 数：0~31 分别指示 1~32 个 Flits。
3	0		
	7	PLENGTH[9:5]	
	6		
	5		
	4		指示尾 DLLDB 中 Payload 终止于哪个 Flit，同时用于计算该 Flit 中 Payload 的大小，CRC 和 Non-CRC 模式下含义不同。
	3		CRC 模式下：
	2		<ol style="list-style-type: none"> <li>当 PLENGTH[4:0] 为 0~15 时，指示 Payload 在 DLLDP 内最后一个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]+1 Bytes；</li> </ol>
	1		<ol style="list-style-type: none"> <li>当 PLENGTH[4:0] 为 16~19 时，指示 Payload 在 DLLDP 内倒数第二个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]+1 Bytes；</li> </ol>
	0		<ol style="list-style-type: none"> <li>当 PLENGTH[4:0] 为 24~27 时，指示 Payload 在 DLLDP 内倒数第二个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]-11 Bytes；</li> <li>PLENGTH[4:0] 为其它，保留。</li> </ol>
			Non-CRC 模式下：
			<ol style="list-style-type: none"> <li>当 PLENGTH[4:0] 为 0~15 时，指示 Payload 在 DLLDP 内最后一个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]+1 Bytes；</li> <li>当 PLENGTH[4:0] 为 17/19 时，指示 Payload 在 DLLDP 内倒数第二个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]+1 Bytes；</li> <li>当 PLENGTH[4:0] 为 27 时，指示 Payload 在</li> </ol>

LPH			
字节编号	bit 编号	域段名	含义
			<p>DLLDP 内倒数第二个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]-11 Bytes；</p> <p>4. 当 PLENGTH[4:0] 为 28~30 时，指示 Payload 在 DLLDP 内最后一个 Flit 位置终结，该 Flit 的 Payload 大小为 PLENGTH[4:0]-11 Bytes；</p> <p>5. PLENGTH[4:0] 为其它，保留。</p>

注 1：每个点到点链路最多支持 16 个 VL；

注 2：由于 BCRC 或 END 位置和长度限制，Payload 可能在最后一个 Flit 或者倒数第二个 Flit 终结，Payload 终结的位置通过 PLENGTH[4:0] 来指示；

注 3：发送端封装 Packet 过程中，识别出最后一个拥有 Payload 的 Flit，然后根据该 Flit 剩余 Payload 的字节数量，计算出 PLENGTH[4:0] 的值，填入 LPH 中。接收端解析 Packet 过程中，通过 PLENGTH[4:0] 计算得到 Payload 的长度，丢弃 Padding。

示例：PLENGTH[4:0] 换算示例

发送端在封装 Packet 时需要计算 PLENGTH[4:0] 并填入域段。

如图 4-7 所示，该 DLLDB 为 DLLDP 中最后一个 DLLDB。Payload 在倒数第二个 Flit 中终结（排除上述 PLENGTH[4:0] 含义中规则 1 的情况），终结位置 Flit 的 Payload 为 16 Bytes，检查 PLENGTH 范围确定为规则 3 的情况，使用公式  $Payload = PLENGTH[4:0] - 11$ ，得  $PLENGTH[4:0] = 27$ ，因此将 PLENGTH[4:0] 域段的值设置为 27。

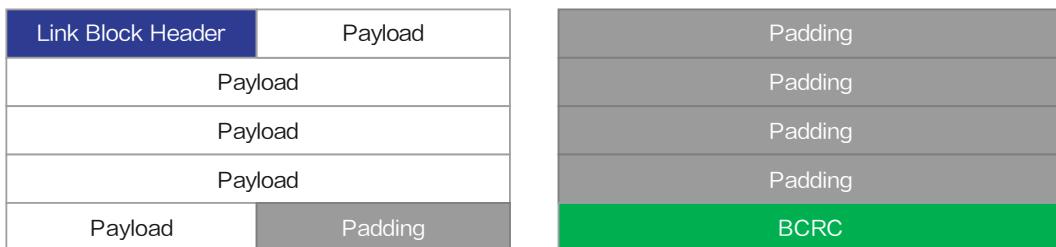


图 4-7 PLENGTH 换算示例 DLLDP/DLLDB 格式 1

如图 4-8 所示，该 DLLDP 中仅有一个 DLLDB。Payload 在倒数第一个 Flit 中终结（只能使用规则 1），终结位置 Flit 的 Payload 为 10 Bytes，根据规则 1，使用公式  $Payload = PLENGTH[4:0] + 1$ ，得  $PLENGTH[4:0] = 9$ ，因此将 PLENGTH[4:0] 域段的值设置为 9。

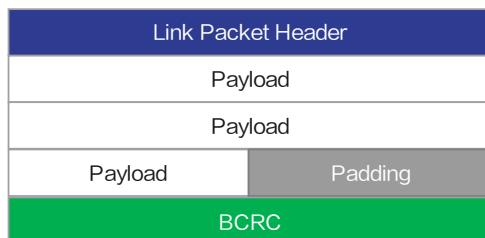


图 4-8 PLENGTH 换算示例 DLLDP/DLLDB 格式 2

如图 4-9 所示，该 DLLDB 为 DLLDP 中最后一个 DLLDB。Payload 在倒数第一个 Flit 中终结（只能使用规则 1），终结位置 Flit 的 Payload 为 14 Bytes，根据规则 1，使用公式  $\text{Payload} = \text{PLENGTH}[4:0] + 1$ ，得  $\text{PLENGTH}[4:0]=13$ ，因此将 PLENGTH[4:0] 域段的值设置为 13。



图 4-9 PLENGTH 换算示例 DLLDP/DLLDB 格式 3

#### 4.3.2.2.3 LBH 格式

LBH ( Link Block Header ) 用于存储 DLLDB 的描述和控制信息，包括是否返还信用证等信息。LBH 域段位置见图 4-5，LBH 位于 Middle Block 和 Last Block 的第一个 Flit，Flit 内部格式见图 4-10，根据 Payload 长度不同，Payload[31:0] 域段也可能为 BCRC。

Byte0								Byte1								Byte2								Byte3							
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
Link Block Header																Payload[143:128]															
Payload[127:96]																															
Payload[95:64]																															
Payload[63:32]																															
Payload[31:0]/BCRC																															

图 4-10 Middle Block 和 Last Block 的第一个 Flit 格式

LBH 共 2 Bytes，其中各 bit 含义见表 4-2。

表 4-2 CRC 模式 LBH 域段

LBH			
字节编号	bit 编号	域段名	含义
0	7	CRD	指示 Block 是否返还信用证，1b'0 表示不返还，1b'1 表示返还，返还粒度在初始化过程中通过协商确定，参数初始化协商相关内容见 4.4 节。若 DLLDP 中多个 CRD 域段置为 1b'1，则表示返还多次信用证。
	6	ACK	指示 Block 是否释放 Retry Buffer 空间，1b'0 表示不释放，1b'1 表示释放，释放粒度在初始化过程中通过协商确定，参数初始化协商相关内容见 4.4 节。若 DLLDP 中多个 ACK 域段置为 1b'1，则表示释放多次 Retry Buffer 空间。

LBH			
字节编号	bit 编号	域段名	含义
1	5	CRD_VL[3:0]	设置为 1b'1, 则表示多次应答并释放 Retry Buffer 空间。
			指示 CRD 域段返还的信用证所对应的 VL 编号。
			保留字段, 发送端默认填 0, 接收端忽略。
1	0	VL[3:0]	Block 的 VL 编号, VL 域段值由网络层携带到数据链路层, VL 相关内容见 4.5 节。
	7		
	6		
	5		
	4		保留字段, 发送端默认填 0, 接收端忽略。
	3	CFG[3:0]	数据链路层通过 CFG 域段区分 DLLCB 和 DLLDP。
	2		CFG == 0 时为 DLLCB, 由数据链路层生成并填充;
	1		CFG == 3/4/5/6/7/9 时为 DLLDP, CFG 域段值由网络层携
	0		带到数据链路层, 具体类型由上层定义, 链路层不感知。 CFG == 其它, 保留。 注: DLLDP 中的 CFG 值必定不为 0。

#### 4.3.2.2.4 BCRC 格式

BCRC 存储了 DLLDB 的 CRC 校验信息, 同时包含 Reserved 和 ERROR\_FLAG 域段, ERROR\_FLAG (仅在 DLLDP 的尾 Flit 有效) 用于指示该 DLLDP 是否包含错误。BCRC 位置见图 4-5, BCRC 域段位于每个 DLLDB 的最后一个 Flit。Flit 内部格式示例见图 4-11。

Byte0								Byte1								Byte2								Byte3																								
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																	
Payload[79:48]																																																
Payload[47:16]																																																
Payload[15:0]																Padding[47:32]																																
Padding[31:0]																																																
BCRC																																																

图 4-11 CRC 模式 DLLDB 的最后一个 Flit 格式

若 DLLDB 只包含 1 个 Flit，则此 Flit 的头部会根据 DLLDB 的位置，携带相应的 LPH 或 LBH 域段。BCRC 域段共 32 bits，其中各 bit 含义见表 4-3。

表 4-3 CRC 模式 BCRC 域段

BCRC			
字节编号	bit编号	域段名	含义
16	7	Reserved	保留字段，发送端默认填0，接收端忽略。
	6	ERROR_FLAG	错误Packet标记，仅在DLLDP的最后一个DLLDB有效： 1'b0：指示此DLLDP为正确的Packet； 1'b1：指示此DLLDP为错误的Packet。 ERROR_FLAG使用方法相关内容见4.8.3节
	5	CRC30[29:0]	该域段为30 bits的CRC计算结果，CRC相关内容见4.7.2节。
	4		
	3		
	2		
	1		
	0		
17:19	7		
	6		
	5		
	4		
	3		
	2		
	1		
	0		

#### 4.3.2.3 Non-CRC 模式

Non-CRC 模式 DLLDP 的 First Block 前 4 字节为 LPH，Middle Block 和 Last Block 的前 2 字节为 LBH，Last Block 的最后一个 Flit 的最后 1 字节为 END 域段。END 域段各 bit 含义见表 4-4，数据链路层通过 END 域段对 Packet 进行标错。Non-CRC 模式下必须启用物理层 FEC 功能，若数据链路层接收到的 Flit 对应的物理层 FEC 解码失败，则通过重传机制进行数据重传。Non-CRC 模式 DLLDP 格式见图 4-12。

## 4 数据链路层

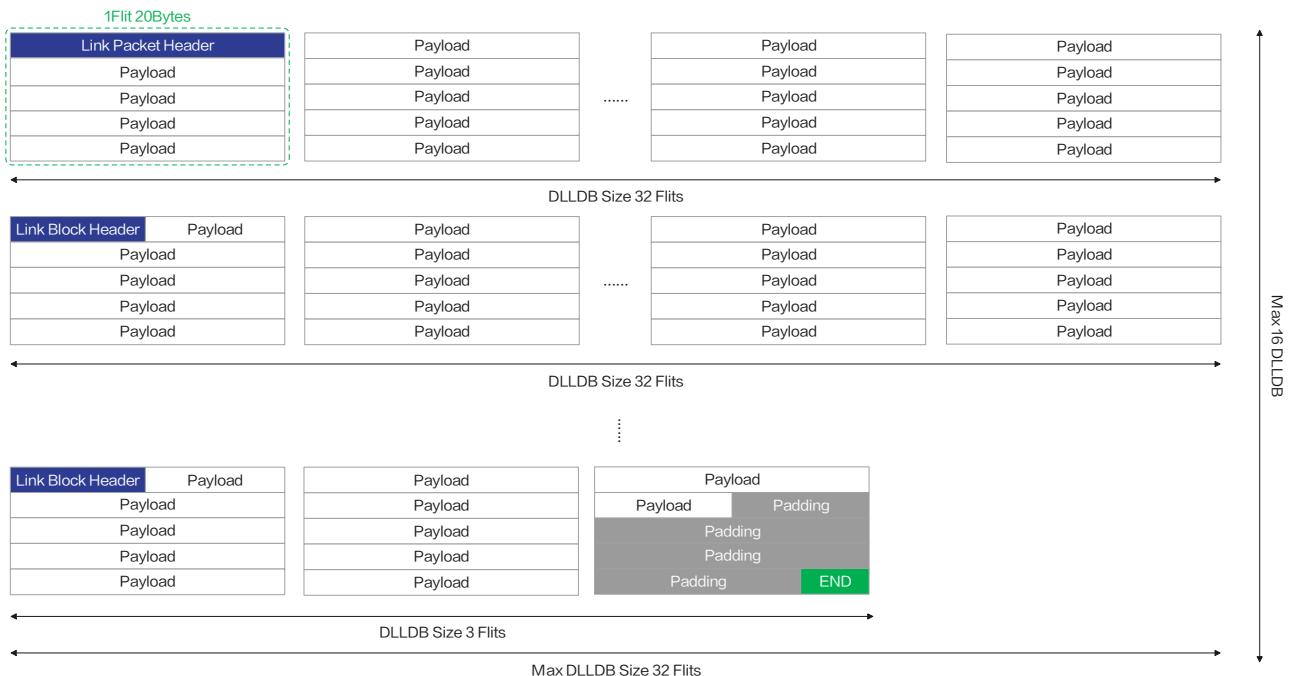


图 4-12 Non-CRC 模式 DLLDP 格式

Non-CRC 模式下的 LPH、LBH 格式与 CRC 模式一致, LPH 格式见 4.3.2.2.2 节,LBH 格式见 4.3.2.2.3 节。见图 4-12, END 域段位于每个 DLLDP 的最后一个 Flit。Flit 格式见图 4-13。如果此 Flit 为 DLLDP/DLLDB 的第一个 Flit, 则携带相应的 LPH/LBH 头部域段。

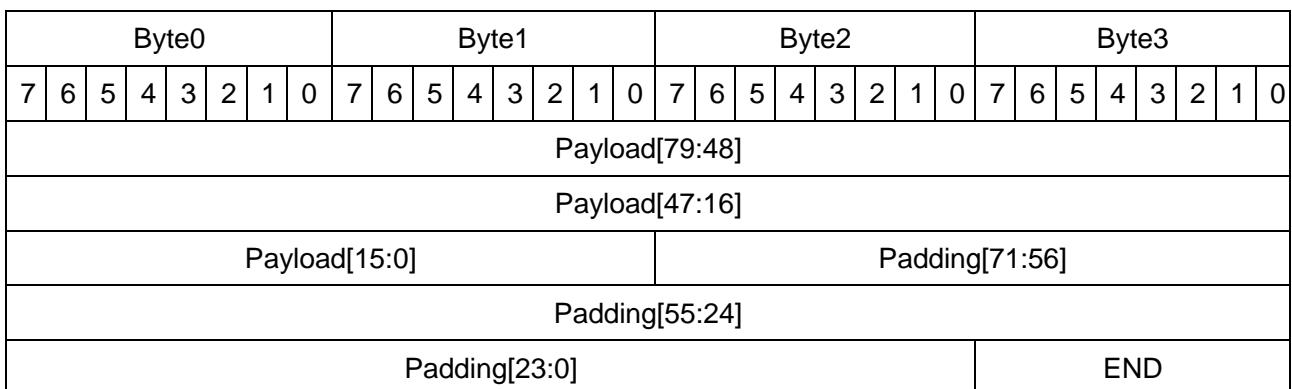


图 4-13 Non-CRC 模式 DLLDP 最后一个 Flit 格式

表 4-4 END 域段

END			
字节编号	bit编号	字段名	含义
19	7	Reserved	保留字段, 发送端默认填0, 接收端忽略。
	6	ERROR_FLAG	错误Packet标记: <ul style="list-style-type: none"> <li>1'b0: 指示此DLLDP为正确的Packet;</li> <li>1'b1: 指示此DLLDP为错误的Packet。</li> </ul>

END			
字节编号	bit编号	字段名	含义
			ERROR_FLAG域段用于标记错误Packet，使用方法见4.8.3节。
	5	Reserved	保留字段，发送端默认填0，接收端忽略。
	4		
	3		
	2		
	1		
	0		

### 4.3.3 DLLCB 格式

#### 4.3.3.1 DLLCB 头部与类型

数据链路层支持收发 DLLCB 以实现信用流控、重传等功能。为避免定界出错，DLLCB 不能插入到 DLLDB 内部，即 DLLCB 只能插入到两个 DLLDB 之间 (Retry Block 除外，重传优先级最高)。DLLCB 组成的最小单元为 Flit，长度范围为 1~32 个 Flits。DLLCB 支持 CRC 模式和 Non-CRC 两种模式，DLLCB 格式见图 4-14 和图 4-15。

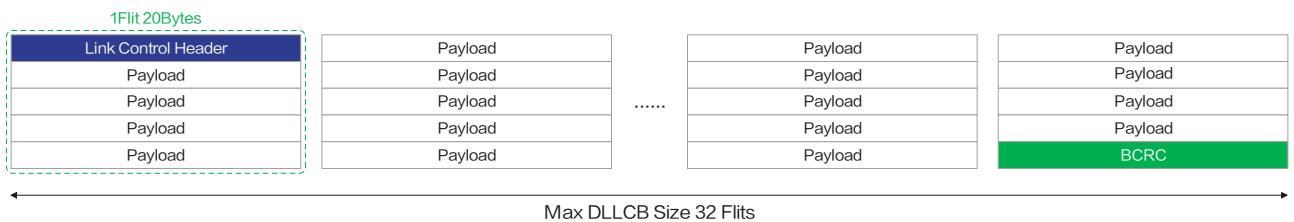


图 4-14 CRC 模式 DLLCB 格式



图 4-15 Non-CRC 模式 DLLCB 格式

LCH ( Link Control Header ) 位于 DLLCB 的头部，字段具体含义见表 4-5。

表 4-5 DLLCB 的 LCH 域段含义

LCH				
字节编号	bit编号	字段名	含义	
0	7	N/A	固定值1'b0。	
	6	CLENGTH	指示DLLCB的Flit长度：0~31分别指示1~32个Flits。	
	5			
	4			
	3			
	2			
	1	N/A	固定值6'b100000。	
	0			
1	7	CFG[3:0]	数据链路层通过CFG域段区分DLLCB和DLLDP。 CFG == 0时为DLLCB，由数据链路层生成并填充； CFG == 3/4/5/6/7/9时为DLLDP，CFG域段值由网络层携带到数据链路层，具体类型由上层定义，链路层不感知。 CFG == 其它，保留。 注：DLLCB中的CFG值必定为0。	
	6			
	5			
	4			
	3			
	2			
	1			
	0			
2	7	CTRL	指示DLLCB类型，具体定义见表 4-6。	
	6			
	5			
	4			
	3	SUB_CTRL	指示DLLCB子类型，具体定义见表 4-6。	
	2			
	1			
	0			
3	7	见4.3.3.2至4.3.3.9具体类型的DLLCB。		
	6			

LCH			
字节编号	bit编号	域段名	含义
5 4 3 2 1 0	5		
	4		
	3		
	2		
	1		
	0		

数据链路层根据 CTRL 和 SUB\_CTRL 域段的值定义了不同种类的 DLLCB，其中 NULL Block/No\_Operation Block/Retry\_Idle Block/Retry\_Req Block/Retry\_Ack Block/Crd\_Ack Block 为数据面 DLLCB，其余 DLLCB 为控制面 DLLCB，见表 4-6。

表 4-6 DLLCB 类型

DLLCB			
DLLCB 类型	CTRL	SUB_CTRL	含义
Null Block	4'b0000	4'b0000	见 4.3.3.2 节
No_Operation Block	4'b0000	4'b0001	见 4.3.3.3 节
Retry_Idle Block	4'b0001	4'b0000	见 4.3.3.4 节
Retry_Req Block	4'b0001	4'b0001	
Retry_Ack Block	4'b0001	4'b0010	
Crd_Ack Block	4'b0010	4'b0100	见 4.3.3.5 节
Param_Exchg Block	4'b0011	4'b0000	见 4.3.3.6 节
Lane_Manage Block	4'b0100	4'b0001	见 4.3.3.7 节
Block_Mode_Chg Block	4'b0101	4'b0000	见 4.3.3.8 节
Init Block	4'b1100	4'b1000	见 4.3.3.9 节

见图 4-14 和图 4-15，在 CRC 和 Non-CRC 两种不同模式下，头部 LCH 域段格式一致，仅尾部有所区别。后续默认以 CRC 模式下的 DLLCB 格式举例，Non-CRC 模式只需将最后一个 Flit 的尾部替换，尾部区别如下：

1. CRC 模式下的 BCRC 域段见 4.3.2.2.4 节；
2. Non-CRC 模式下的 END 域段见图 4-13 和表 4-4 描述。

示例：优先级设置示例

数据链路层发送端发送 DLLCB/DLLDP 时优先级可根据实现调整，此处供参考。可设置优先级从高到低分别为：

Retry\_Ack\_Set ( 包括连续的 1 个 Retry\_Idle Block 和 32 个 Retry\_Ack Block ) > Retry\_Req\_Set ( 包括连续的 1 个 Retry\_Idle Block 和 32 个 Retry\_Req Block ) > Block\_Mode\_Chg Block > Crd\_Ack Block ( Type == 1'b0 ) > Lane\_Manage Block > Param\_Exchg Block > Crd\_Ack Block ( Type == 1'b1 ) > Init Block > No\_Operation Block > DLLDP > Null Block

### 4.3.3.2 Null Block

Null Block 是在数据链路层无 DLLDP 传输时，即链路空闲时发送的 Block，对端接收到 Null Block 后直接将其丢弃。Null Block 的发送和丢弃可以实现在链路层或物理层，与实现相关，此处不做限定。Null Block 长度为 1 Flit，格式见图 4-16。

Byte0								Byte1								Byte2								Byte3																															
1'b0		CLENGTH		6'b100000				CFG		CTRL				SUB_CTRL				Reserved								Reserved																													
Reserved																																																							
Reserved																																																							
Reserved																																																							
BCRC																																																							

图 4-16 CRC 模式 Null Block 格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00000，表示 Null Block 长度为 1 Flit；
2. CFG: 值为 4'b0000，表示 Block 为 DLLCB 类型；
3. CTRL 值为 4'b0000，SUB\_CTRL 值为 4'b0000，表示 DLLCB 类型为 Null Block，其它类型见表 4-6。

### 4.3.3.3 No\_Operation Block

No\_Operation Block ( NOB ) 是在发送小包（这里的小包指长度小于 PACKET\_MIN\_INTERVAL 的 DLLDP 包）时填充的 Block，由于接收端小包处理速率有限，需要发送端降低小包发送速率，否则会出现流量激增导致接收端无法处理的情况。小包大小可根据实际业务场景定义。发送方在小包后面紧跟着填充若干 NOB，以满足 Init Block 中的 PACKET\_MIN\_INTERVAL 域段规定的连续两个 DLLDP 间的最小间隔；接收方数据链路层收到 NOB 后将其丢弃。NOB 发送时会在本端的 Retry Buffer 中保存，长度为 1 Flit，格式见图 4-17：

Byte0								Byte1								Byte2								Byte3																																								
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																																	
1'b0	CLENGTH				6'b100000								CFG				CTRL				SUB_CTRL				Reserved																																							
Reserved																																																																
Reserved																																																																
Reserved																																																																
BCRC																																																																

图 4-17 CRC 模式 NOB 格式

其中各域段含义为：

1. CLENGTH 值为 5'b00000，表示 NOB 长度为 1 Flit；
2. CFG 值为 4'b0000，表示 Block 为 DLLCB 类型；
3. CTRL 值为 4'b0000，SUB\_CTRL 值为 4'b0001，表示 DLLCB 类型为 NOB。

#### 4.3.3.4 Retry Block

##### 4.3.3.4.1 Retry\_Idle Block

Retry\_Idle Block 用于隔离 DLLDP 和 Retry\_Req/Retry\_Ack Block，防止接收端误将 Payload 识别为 Retry\_Req/Retry\_Ack Block 导致处理错误。Retry\_Idle Block 长度为 1 Flit。

Byte0								Byte1								Byte2								Byte3																																								
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																																	
1'b0	CLENGTH				6'b100000								CFG				CTRL				SUB_CTRL				Reserved																																							
Reserved																																																																
Reserved																																																																
Reserved																																																																
BCRC																																																																

图 4-18 CRC 模式 Retry\_Idle Block 格式

其中各域段含义为：

1. CLENGTH：值为 5'b00000，表示 Retry\_Idle Block 长度为 1 Flit；
2. CFG：值为 4'b0000，表示 Block 为 DLLCB 类型；
3. CTRL 值为 4'b0001，SUB\_CTRL 值为 4'b0000，表示 DLLCB 类型为 Retry\_Idle Block，其它类型见表 4-6。

#### 4.3.3.4.2 Retry\_Req Block

Retry\_Req Block 用于发出重传请求，长度为 1 Flit。

Byte0								Byte1								Byte2								Byte3																			
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												
1'b0	CLENGTH				6'b100000				CFG				CTRL				SUB_CTRL				Reserved																						
Reserved																																											
RcvPtr								NUM_PHY_REINIT								NUM_RETRY								Reserved																			
BCRC																																											

图 4-19 CRC 模式 Retry\_Req Block 格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00000, 表示 Retry\_Req Block 长度为 1 Flit;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b0001, SUB\_CTRL 值为 4'b0001, 表示 DLLCB 类型为 Retry\_Req Block, 其它类型见表 4-6;
4. RcvPtr: 重传的起始地址, 即待重传的第一个 Flit 在对端 Retry Buffer 中的编号;
5. NUM\_RETRY: 一次重传事件内重初始化物理层之前发起重传请求序列的次数, 重初始化物理层后清 0, 重新计数;
6. NUM\_PHY\_REINIT: 一次重传事件内重初始化物理层的次数。

#### 4.3.3.4.3 Retry\_Ack Block

Retry\_Ack Block 用于向对端返回重传应答, 长度为 1 Flit。

Byte0								Byte1								Byte2								Byte3																			
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												
1'b0	CLENGTH				6'b100000				CFG				CTRL				SUB_CTRL				Reserved																						
Reserved																																											
NumFreeBuf								RdPtr								WrPtr								Reserved																			
BCRC																																											

图 4-20 CRC 模式 Retry\_Ack Block 格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00000, 表示 Retry\_Ack Block 长度为 1 Flit;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b0001, SUB\_CTRL 值为 4'b0010, 表示 DLLCB 类型为 Retry\_Ack Block, 其它类型见表 4-6;
4. NUM\_RETRY: 一次重传事件内重初始化物理层之前发起重传请求序列的次数, 重初始化物理层后清 0, 重新计数;
5. NumFreeBuf: Retry Buffer 的空闲空间, 单位为 Flit;
6. RdPtr: 本次重传事件的 Retry Buffer 读指针;
7. WrPtr: 本次重传事件的 Retry Buffer 写指针。

#### 4.3.3.5 Crd\_Ack Block

数据链路层支持基于 VL 的信用流控, 数据链路层可以通过发送 Crd\_Ack Block 给对端返还信用证, 信用流控相关内容见 4.6 节。

数据链路层支持重传功能, 其中本端数据链路层可以通过发送 Crd\_Ack Block 给对端, 对 Retry Buffer 中的 Flit 进行应答, 从而释放对端的 Retry Buffer 空间, Retry Buffer 管理相关内容见 4.7.3.2 节。

Crd\_Ack Block 长度为 2 Flits, 格式如下图 4-21 和图 4-22。

Byte0										Byte1										Byte2										Byte3																																				
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	7	6	5	4	3	2	1	0																											
1'b0	CLENGTH			6'b100000					CFG			CTRL			SUB_CTRL			S	Reserved					T																																										
ACK_NUM																				CRD_NUM[95:80]																																														
CRD_NUM[79:48]																																																																		
CRD_NUM[47:16]																																																																		
CRD_NUM[15:0]										Reserved																																																								

图 4-21 CRC 模式 Crd\_Ack Block Flit0 格式

Byte0										Byte1										Byte2										Byte3									
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	7	6	5	4	3	2	1	0
Reserved																																							
Reserved																																							
Reserved																																							
BCRC																																							

图 4-22 CRC 模式 Crd\_Ack Block Flit1 格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00001, 表示 Crd\_Ack Block 长度为 2 Flits, Non-CRC 模式下长度保持一致, 不会将 END 域段填到第一个 Flit;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b0010, SUB\_CTRL 值为 4'b0100, 表示 DLLCB 类型为 Crd\_Ack Block, 其它类型见表 4-6;
4. SD: SEND\_DONE, 初始化信用证发送完成标志, 1'b1 表示初始化信用证发送完成; 1'b0 表示初始化信用证发送未完成。只有在 Type==1'b1 时此域段有效, Type 域段定义见下文 T 域段的描述;
5. T: Type, 信用证类型。1'b1 表示携带的是初始化信用证; 1'b0 表示返还的是非初始化信用证。初始化阶段, 可能需要发送多个 Crd\_Ack Block, 当 SEND\_DONE==1'b1 且 Type==1'b1 时, 指示发送的是最后一个 Crd\_Ack Block;
6. ACK\_NUM: 位宽 16 bits, 指示 Retry Buffer 空间释放数量。ACK\_NUM 对应粒度由 CTRL\_ACK\_GRAIN\_SIZE 指定, 释放空间大小为 ACK\_NUM\*CTRL\_ACK\_GRAIN\_SIZE 个 Flits, CTRL\_ACK\_GRAIN\_SIZE 大小在初始化中协商, 参数初始化协商相关内容见 4.4 节;
7. CRD\_NUM: 共 96 bits, 每 6 bits 表示一个 VL 携带或返还的信用证数量, 16 个 VL 携带或返还的信用证数量分别为 {Crd\_num\_VL15, Crd\_num\_VL14, ……, Crd\_num\_VL0}。信用证粒度由 CTRL\_CREDIT\_GRAIN\_SIZE 指定, 每个 VL 携带或返还信用证数量为 Crd\_num\_VL(N)\* CTRL\_CREDIT\_GRAIN\_SIZE(N) 个 Cells, CTRL\_CREDIT\_GRAIN\_SIZE 大小在初始化中协商, N 表示对应的 VL 编号, Cell 为信用流控的基本单元, 参数初始化协商相关内容见 4.4 节, 信用证初始化相关内容见 4.6.1.1 节。

#### 4.3.3.6 Param\_Exchg Block

此 Block 专门为网络层提供服务, 用于交换邻居通告信息。当数据链路层进入 DLL\_Normal 状态后, 如果本端设备仍然需要和对端设备进行信息交互, 数据链路层支持通过传输 Param\_Exchg Block 来进行信息交互。Param\_Exchg Block 长度为 1~32 个 Flits, 假设某 Param\_Exchg Block 的 Flit 数为 N, 格式如下图 4-23 到图 4-25。

Byte0								Byte1								Byte2								Byte3																																	
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																										
1'b0	CLENGTH				6'b100000				CFG				CTRL				SUB_CTRL				Reserved																																				
Reserved																																																									
PARAM_EXCHG_DATA[31:0]																																																									
Reserved																																																									
Reserved																																																									

图 4-23 CRC 模式 Param\_Exchg Block 格式 Flit0

Byte0								Byte1								Byte2								Byte3								
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	
Reserved																																
Reserved																																
PARAM_EXCHG_DATA[31:0]																																
Reserved																																
Reserved																																

图 4-24 CRC 模式 Param\_Exchg Block 格式 Flit1~Flit(N-2)

Byte0								Byte1								Byte2								Byte3								
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	
Reserved																																
Reserved																																
PARAM_EXCHG_DATA[31:0]																																
Reserved																																
BCRC																																

图 4-25 CRC 模式 Param\_Exchg Block 格式 Flit(N-1)

其中各域段含义为：

1. CLENGTH: 值为 5'b00000~5'b11111，表示 Param\_Exchg Block 长度为 1~32 个 Flits；
2. CFG: 值为 4'b0000，表示 Block 为 DLLCB 类型；
3. CTRL 值为 4'b0011，SUB\_CTRL 值为 4'b0000，表示 DLLCB 类型为 Param\_Exchg Block，其它类型见表 4-6；
4. PARAM\_EXCHG\_DATA: 表示邻居通告信息，相关内容见 10.4.4 节；

#### 4.3.3.7 Lane\_Manage Block

UB 物理层支持动态升降 Lane 功能：本端物理层向本端数据链路层发起 Lane 切换请求，触发本端数据链路层通过 Lane\_Manage Block ( LM Block ) 给两端的物理层传输动态升降 Lane 命令，详细流程见 3.4.2.3 节。Lane\_Manage Block 长度为 1 Flit，格式如下图 4-26。

Byte0								Byte1								Byte2								Byte3								
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	
01	CLLENGTH				6'b100000				CFG				CTRL				SUB_CTRL				Reserved											
	CMD[63:32]																															

Byte0								Byte1								Byte2								Byte3							
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
CMD[31:0]																Reserved															
BCRC																															

图 4-26 CRC 模式 Lane\_Manage Block 格式

其中各域段含义为：

1. LENGTH: 值为 5'b00000, 表示 Lane\_Manage Block 长度为 1 Flit;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b0100, SUB\_CTRL 值为 4'b0001, 表示 DLLCB 类型为 Lane\_Manage Block, 其它类型见表 4-6;
4. CMD 域段内容见表 4-7。

表 4-7 CMD 域段

CMD			
字节编号	bit编号	域段名	含义
0	7	REQ_ID	切换Link Width请求标识, 固定值为0x5。
	6		
	5		
	4		
	3	Reserved	保留字段, 发送端默认填0, 接收端忽略。
	2	Power_Down	请求关闭所有Lane, 1b'1有效。
	1	Change_RLW	请求切换RLW ( RX_Link_Width ), 1b'1有效。
	0	Change_TLW	请求切换TLW ( TX_Link_Width ), 1b'1有效。
1	7	RSP_ID	响应标识, 固定值为0xA。
	6		
	5		
	4		
	3	Reserved	保留字段, 发送端默认填0, 接收端忽略。
	2	RX_Lane_Up	用于应答升Lane完成, 1'b1表示升Lane完成。
	1	RSP_RLW	切换RLW请求的响应, 1b'0NAK, 1b'1ACK。
	0	RSP_TLW	切换TLW请求的响应, 1b'0NAK, 1b'1ACK。

CMD			
字节编号	bit编号	域段名	含义
2	7	Reserved	保留字段，发送端默认填0，接收端忽略。
	6		
	5	TLW	6'b000001: TLW值为X1; 6'b000010: TLW值为X2; 6'b000100: TLW值为X4; 6'b001000: TLW值为X8; 其它值: 保留。
	4		
	3		
	2		
	1		
	0		
3	7	Reserved	保留字段，发送端默认填0，接收端忽略。
	6		
	5	RLW	6'b000001: RLW值为X1; 6'b000010: RLW值为X2; 6'b000100: RLW值为X4; 6'b001000: RLW值为X8; 其它值: 保留。
	4		
	3		
	2		
	1		
	0		
4:7	7	Reserved	保留字段，发送端默认填0，接收端忽略。
	6		
	5		
	4		
	3		
	2		
	1		
	0		

#### 4.3.3.8 Block\_Mode\_Chg Block

UB 支持数据链路层封装模式切换功能，可以在 CRC 与 Non-CRC 模式之间进行切换，数据链路层负责通过 Block\_Mode\_Chg Block 来传输切换命令给两端的数据链路。Block\_Mode\_Chg Block 长度为 1 Flit，格式如下图 4-27。

Byte0								Byte1								Byte2								Byte3																				
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													
15	CLENGTH			6'b100000					CFG			CTRL		SUB_CTRL			Reserved			TYPE		Reserved																						
Reserved								Reserved								Reserved								BCRC																				

图 4-27 CRC 模式 Block\_Mode\_Chg Block 格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00000, 表示 Block\_Mode\_Chg Block 长度为 1Flit;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b0101, SUB\_CTRL 值为 4'b0000, 表示 DLLCB 类型为 Block\_Mode\_Chg Block, 其它类型见表 4-6;
4. TYPE: 4'b0000 表示当前 Block\_Mode\_Chg Block 为 Block\_Mode\_Chg\_REQ 类型; 4'b0001 表示当前 Block\_Mode\_Chg Block 为 Block\_Mode\_Chg\_ACK 类型。

示例：封装模式切换示例

通过 Block\_Mode\_Chg Block 切换 CRC 与 Non-CRC 封装模式的流程，如图 4-28 所示，具体步骤如下：

1. 封装模式切换有两种来源：一是本端软件需要切换封装模式，通知物理层切换，然后物理层发起封装模式切换；二是物理层根据 BER，主动发起封装模式切换。两种来源均由物理层 Primary Port 发起封装模式切换，触发数据链路层配合完成，物理层 Primary Port 相关定义见 3.4.2.1 节；
2. 本端数据链路层接收到来自物理层的模式切换请求后，反压网络层流量；
3. 反压完成后，发起封装模式切换流程，即向对端发送 Block\_Mode\_Chg\_REQ，此后本端数据链路层只能发送 Crd\_Ack Block 或 Block\_Mode\_Chg\_ACK；
4. 对端数据链路层收到 Block\_Mode\_Chg\_REQ 之后，也反压网络层流量；
5. 反压完成后，对端发送 Block\_Mode\_Chg\_REQ 给本端数据链路层，告知本端 Block\_Mode\_Chg\_REQ 已收到，此后对端数据链路层只能发送 Crd\_Ack Block 或 Block\_Mode\_Chg\_ACK；
6. 本端数据链路层收到 Block\_Mode\_Chg\_REQ 后，将本地所有待返回的 ACK 通过 Crd\_Ack Block 返回到对端，并发送 Block\_Mode\_Chg\_ACK 给对端数据链路层，告知对端，本端已收到对端的 Block\_Mode\_Chg\_REQ，此后本端数据链路层不发送任何 DLLCB/DLLDP；
7. 对端数据链路层收到 Block\_Mode\_Chg\_ACK 后，将本地所有待返回的 ACK 通过 Crd\_Ack Block 返回到本端，并发送 Block\_Mode\_Chg\_ACK 给本端数据链路层，告知本端，对端已收到本端的 Block\_Mode\_Chg\_ACK，此后对端数据链路层不发送任何 DLLCB/DLLDP；

8. 本端数据链路层收到 Block\_Mode\_Chg\_ACK 后,发送模式切换响应给本端物理层,触发本端物理层进行 Retrain, 同时发送 RLTB ( Retrain Link Training Block ) 给对端物理层, 对端物理层收到 RLTB 后, 开始 Retrain, RLTB 相关内容见 3.4.1.1 节;
9. 本端物理层 Retrain 完成后, 发送新模式启用信号给本端数据链路层, 对端也执行对应操作;
10. 本端数据链路层解除对网络层的流量反压, 对端也执行对应操作;
11. 至此, 整个封装模式切换流程完成, 链路两端开始用切换后的封装模式进行通信。

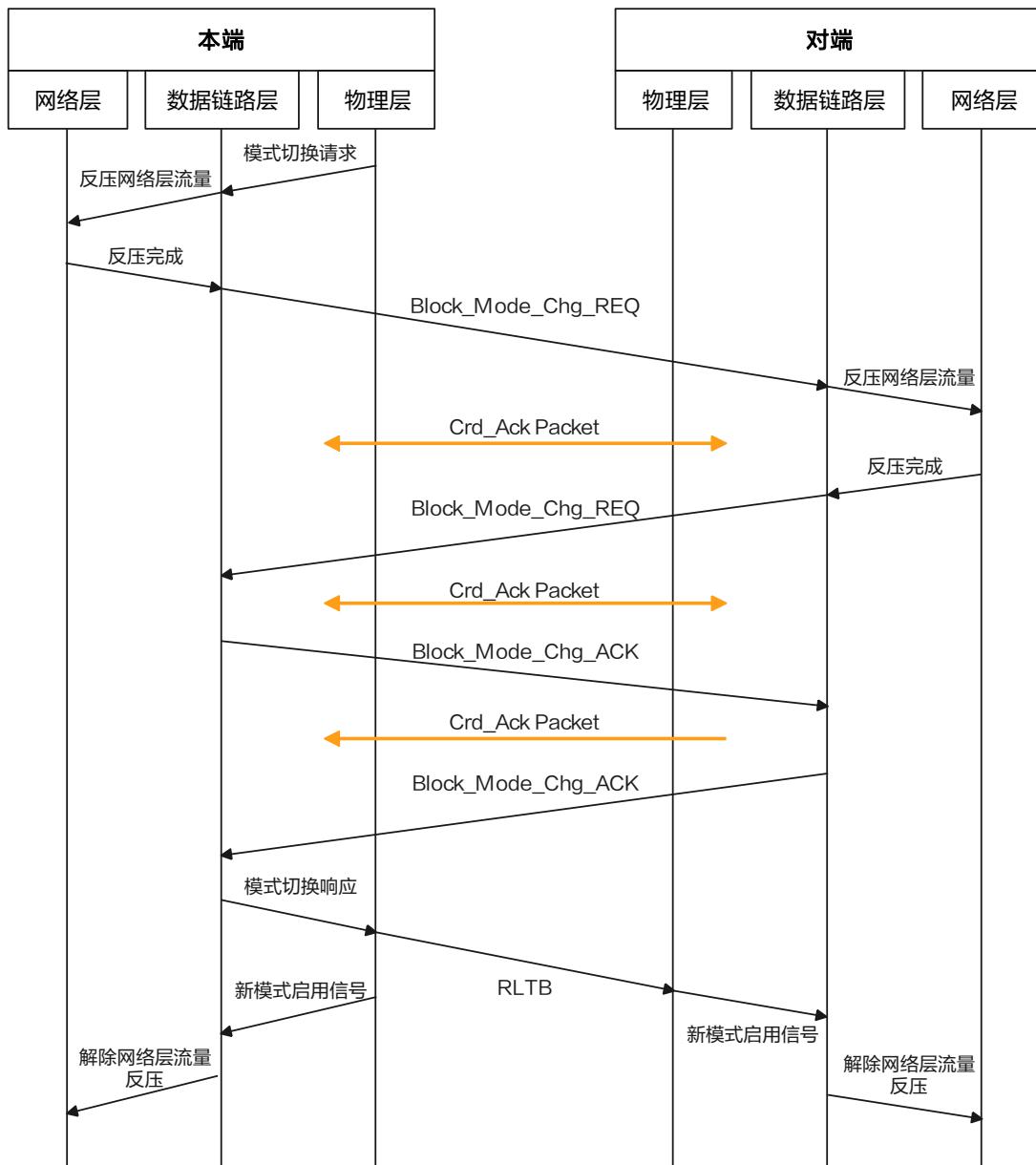


图 4-28 封装模式切换流程

#### 4.3.3.9 Init Block

数据链路层参数初始化阶段需要收发 Init Block, 实现以下功能:

1. 在 DLL\_Param\_Init 状态下，两端数据链路层互相收发 Init Block 完成握手，从而确保数据链路层通信正常；
2. 在 DLL\_Param\_Init 状态下，数据链路层发送 Init Block，Init Block 中携带初始化参数告知对端数据链路层本端配置的工作模式，从而完成自协商。

Init Block 的长度 N 可变，N 可以为 5~32 个 Flits，前 4 个 Flits 是基本的初始化参数，最后一个 Flit 用于存放 BCRC/END，因此 N 至少为 5。多余的 Flits 可在扩展初始化参数时使用。Init Block 格式如下图 4-29 至图 4-34。

Byte0								Byte1								Byte2								Byte3															
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								
0 1	CLENGTH				6'b100000				CFG				CTRL				SUB_CTRL				Reserved																		
Reserved								FEATURE_ID[15:8]								FEATURE_ID[7:0]								Reserved															
Reserved								R				DATA_ACK_GRAIN_SIZE				CTRL_ACK_GRAIN_SIZE				FLOW_CTRL_SIZE								VL_ENABLE				Reserved							
RETRY_BUF_DEPTH								Reserved								Reserved								Reserved															

图 4-29 CRC 模式 Init Block Flit0 格式

Byte0								Byte1								Byte2								Byte3							
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
Reserved								CTRL_CREDIT_GRAIN_SIZE[111:80]								CTRL_CREDIT_GRAIN_SIZE[79:48]								CTRL_CREDIT_GRAIN_SIZE[47:16]							
CTRL_CREDIT_GRAIN_SIZE[15:0]								Reserved								Reserved								Reserved							

图 4-30 CRC 模式 Init Block Flit1 格式

Byte0								Byte1								Byte2								Byte3							
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
Reserved								DATA_CREDIT_GRAIN_SIZE[95:64]								DATA_CREDIT_GRAIN_SIZE[63:32]								DATA_CREDIT_GRAIN_SIZE[31:0]							
CTRL_CREDIT_GRAIN_SIZE[127:112]								Reserved								Reserved								Reserved							

图 4-31 CRC 模式 Init Block Flit2 格式

Byte0								Byte1								Byte2								Byte3							
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
Reserved																															
Reserved																															
Reserved								PACKET_MIN_INTERVAL								DATA_CREDIT_GRAIN_SIZE[127:112]															
DATA_CREDIT_GRAIN_SIZE[111:96]																Reserved															

图 4-32 CRC 模式 Init Block Flit3 格式

Byte0								Byte1								Byte2								Byte3							
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
Reserved																															
Reserved																															
Reserved																															
Reserved																															

图 4-33 CRC 模式 Init Block Flit4~Flit(N-2)格式

Byte0								Byte1								Byte2								Byte3							
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
Reserved																															
Reserved																															
Reserved																															
BCRC																															

图 4-34 CRC 模式 Init Block Flit(N-1)格式

其中各域段含义为：

1. CLENGTH: 值为 5'b00100~5'b11111, 表示 Init Block 长度为 5~32 Flits;
2. CFG: 值为 4'b0000, 表示 Block 为 DLLCB 类型;
3. CTRL 值为 4'b1100, SUB\_CTRL 值为 4'b1000, 表示 DLLCB 类型为 Init Block, 其它类型见表 4-6;
4. R: RXBUF\_VL\_SHARE 域段解释见表 4-8。

Init Block 中 Flit0~Flit3 携带了初始化参数内容, 所有初始化参数内容见表 4-8。

表 4-8 Init Block 域段信息

Init Block			
Flit号	bit编号	域段名	含义
0	111:88	Reserved	保留字段，发送端默认填0，接收端忽略。
	87:72	FEATURE_ID	数据链路层协议版本号，本规范支持的值为1。
	71:65	Reserved	保留字段，发送端默认填0，接收端忽略。
	64	RXBUF_VL_SHARE	Receive Buffer是否支持VL共享，1'b1：支持；1'b0：不支持。
	63:56	DATA_ACK_GRAIN_SIZE	DLLDP支持通过LPH/LBH的ACK域段释放Retry Buffer空间，释放的粒度通过DATA_ACK_GRAIN_SIZE协商确定（单位：Flit）。每bit对应一个Flit数量，分别为{128,64,32,16,8,4,2,1}。
	55:48	CTRL_ACK_GRAIN_SIZE	Crd_Ack Block支持通过ACK_NUM域段释放Retry Buffer空间，释放粒度通过CTRL_ACK_GRAIN_SIZE协商确定（单位：Flit）。每bit对应一个Flit数量，分别为{128,64,32,16,8,4,2,1}。
	47:40	FLOW_CTRL_SIZE	支持信用流控最小单元Cell对应的Flit数量：每bit对应一个Flit数量，分别为{128,64,32,16,8,4,2,1}。
	39:24	VL_ENABLE	支持启用的最多16个虚拟通道，bit0~bit15分别对应VL0~VL15，必须启用VL0。1'b1：表示支持启用此VL；1'b0：表示不支持启用此VL。
	23:16	Reserved	保留字段，发送端默认填0，接收端忽略。
	15:0	RETRY_BUF_DEPTH	Retry Buffer深度，单位：Flit。（该域段不需要进行协商，直接设置为接收到的数值即可）
1	159:112	Reserved	保留字段，发送端默认填0，接收端忽略。
	111:0	CTRL_CREDIT_GRAIN_SIZE[111:0]	Crd_Ack Block支持通过CRD_NUM域段返还各个VL返还信用证，其中VL0~VL13的信用证返还粒度通过CTRL_CREDIT_GRAIN_SIZE[111:0]协商确定（单位Cell）。每个Byte对应一个VL的信用证返还粒度{CTRL_CREDIT_GRAIN_SIZE_VL13, CTRL_CREDIT_GRAIN_SIZE_VL12,……, CTRL_CREDIT_GRAIN_SIZE_VL0}。Byte内每bit对应一个Cell数量，分别为{128,64,32,16,8,4,2,1}。
2	159:112	Reserved	保留字段，发送端默认填0，接收端忽略。
	111:16	DATA_CREDIT_GRAIN_SIZE[95:0]	DLLDP支持通过LPH/LBH的CRD域段返还某个VL的信用证，其中VL0~VL11的信用证返还粒度通过

Init Block			
Flit号	bit编号	域段名	含义
			DATA_CREDIT_GRAIN_SIZE[95:0]协商确定（单位Cell）。每个Byte对应一个VL的信用证返还粒度{DATA_CREDIT_GRAIN_SIZE_VL11, DATA_CREDIT_GRAIN_SIZE_VL10,……, DATA_CREDIT_GRAIN_SIZE_VL0}。Byte内每bit对应一个Cell数量，分别为{128,64,32,16,8,4,2,1}。
			15:0 CTRL_CREDIT_GRAIN_SIZE[127:112] Crd_Ack Block支持通过CRD_NUM域段返还各个VL的信用证，其中VL14~VL15的信用证返还粒度通过CTRL_CREDIT_GRAIN_SIZE[127:112]协商确定（单位Cell）。每个Byte对应一个VL的信用证返还粒度{CTRL_CREDIT_GRAIN_SIZE_VL15, CTRL_CREDIT_GRAIN_SIZE_VL14}。Byte内每bit对应一个Cell数量，分别为{128,64,32,16,8,4,2,1}。
3	111:40	Reserved	保留字段，发送端默认填0，接收端忽略。
	39:32	PACKET_MIN_INTERVAL	接收端支持接收到的连续两个DLLDP间的最小间隔，该域段的值即为最小间隔值，指当前DLLDP的首Flit到下一个DLLDP首Flit之间的间隔（不包含下一个DLLDP的首Flit）。单位：Flit。（该域段不需要进行协商，直接设置为接收到的数值即可）
	31:0	DATA_CREDIT_GRAIN_SIZE[127:96]	DLLDP支持通过LPH/LBH的CRD域段返还某个VL的信用证，其中VL12~VL15的信用证返还粒度通过DATA_CREDIT_GRAIN_SIZE[127:96]协商确定（单位Cell）。每个byte对应一个VL的信用证返还粒度{DATA_CREDIT_GRAIN_SIZE_VL15, DATA_CREDIT_GRAIN_SIZE_VL14, DATA_CREDIT_GRAIN_SIZE_VL13, DATA_CREDIT_GRAIN_SIZE_VL12}。Byte内每bit对应一个Cell数量，分别为{128,64,32,16,8,4,2,1}。
4~N-2	159:0	Reserved	保留字段，发送端默认填0，接收端忽略。
N-1	159:32	Reserved	保留字段，发送端默认填0，接收端忽略。

注：协议中定义的初始化参数内容必须在 Init Block 中发给对端链路层，不允许只发送部分初始化参数内容。

Init Block 参与的初始化流程和自协商过程见 4.4 节。

### 4.3.4 DLLCB/DLLDP 收发流程

#### 4.3.4.1 DLLCB/DLLDP 定界

定界发生在 DLLCB/DLLDP 接收端，物理层每次以 Flit 粒度向数据链路层提交收到的数据。数据链路层收到 Flit 后，进行 DLLCB/DLLDP 边界定位。CRC 模式与 Non-CRC 模式的定界流程不同，具体说明如下：

CRC 模式的 DLLCB/DLLDP 定界流程见图 4-35：

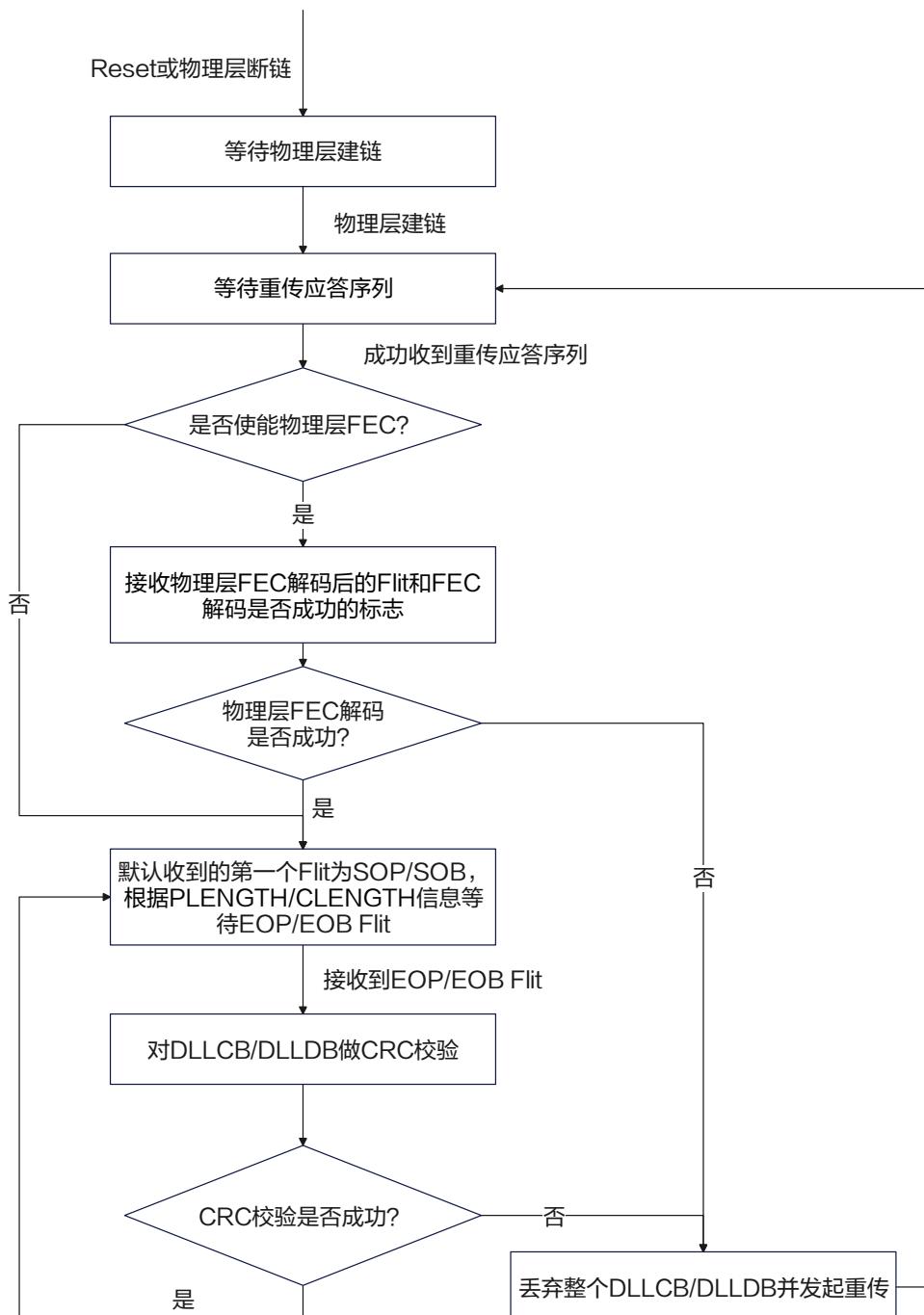


图 4-35 CRC 模式定界流程

其中：

1. SOP: Start Of Packet, DLLDP 的首个 Flit;
2. SOB: Start Of Block, DLLCB/DLLDB 的首个 Flit;
3. EOP: End Of Packet, DLLDP 的最后一个 Flit;
4. EOB: End Of Block, DLLCB/DLLDB 的最后一个 Flit。

需要注意：

1. 为保证定界成功，对发送端有以下要求：
  - (1) DLLDP 内的 DLLDB 之间不能插入其它 DLLDP 的 DLLDB；
  - (2) DLLCB 仅能在 DLLDB 间插入。
2. 当 FEC 解码失败（启用物理层 FEC 时）或 CRC 校验失败后数据链路层会发起重传，告知对端重新发送此 DLLDB，未收到 Retry\_Ack Block 前接收到的除 Retry\_Idle Block、Retry\_Req Block 外的所有 DLLCB/DLLDP 均丢弃；
3. 物理层建链完成后，数据链路层首先发起重传流程：发送端发起重传请求序列，直到收到重传响应序列。发起重传流程是为了防止物理层建链时间不一致造成 DLLCB 丢失，导致数据链路层建链失败。该重传流程无特殊要求，按正常重传流程进行即可。重传流程和重传序列相关内容见 4.7 节。

Non-CRC 模式的 DLLCB/DLLDP 定界流程见图 4-36。Non-CRC 模式下 Block 中不包含 CRC 域段，因此其重传触发条件有所不同。当 FEC 解码失败后数据链路层会发起重传，丢弃此 FEC 帧对应的 Flit 及之后的所有 Flit，告知对端从此 FEC 帧对应的 Flit 开始重传，未收到重传应答序列前接收到的其它 DLLCB/DLLDP 均丢弃，注意事项与 CRC 模式类似，重传应答序列相关内容见 4.7.3.4 节。

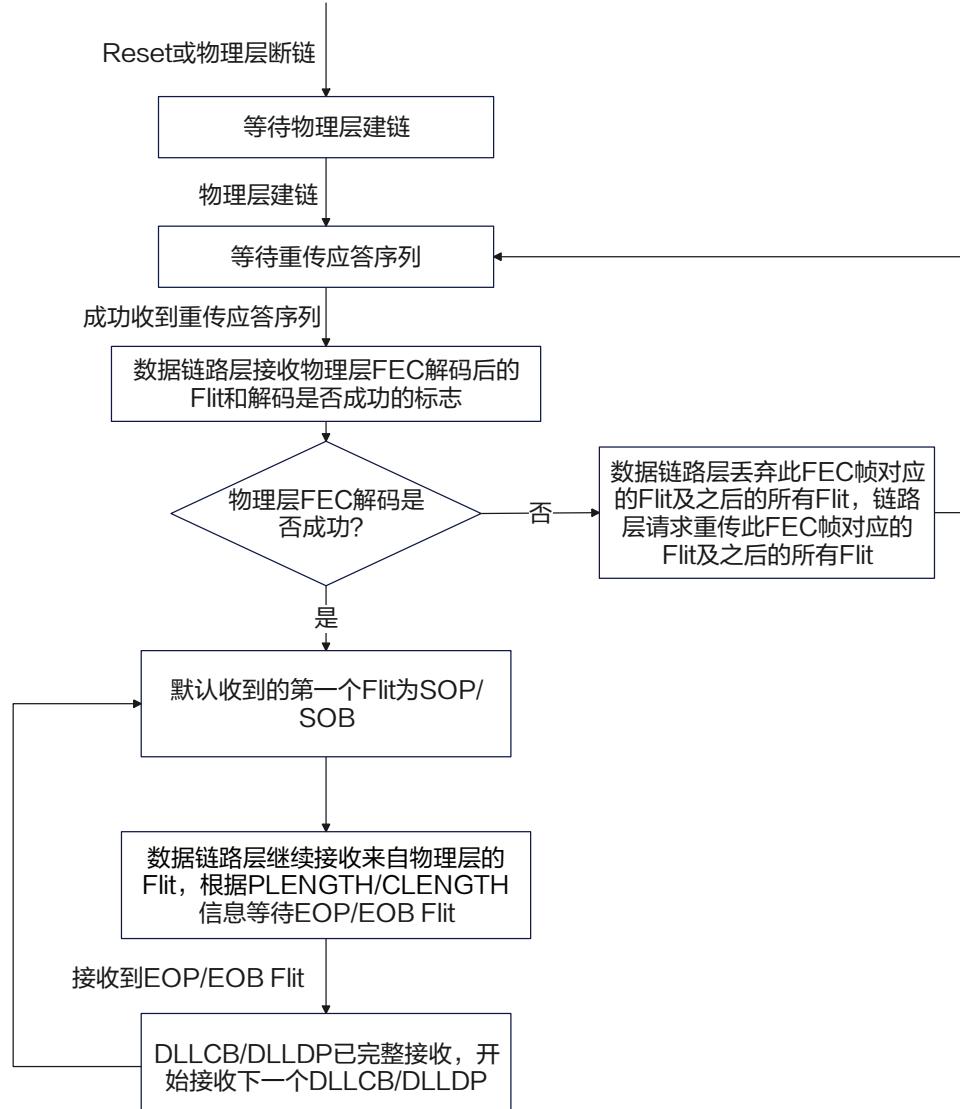


图 4-36 Non-CRC 模式定界流程

#### 4.3.4.2 发送流程

在发送端，数据链路层将网络层下发的 Packet 作为数据链路层的 Payload，并封装成 DLLDP。DLLDP 封装是在 Payload 的基础上添加头部 LPH/LBH、Padding(可选)、尾部 BCRC/END，然后按头部、Payload、Padding(可选)、尾部的顺序填充到 Flit 中，具体格式见 4.3.2 节。

Flit 填充规则如下：从首个 Flit 开始，按 Flit 顺序逐一填充满 Flit。若最后一个 Flit 无法用 Payload 填充满 20 Bytes，则使用 Padding 填充剩余位置，Padding 值统一设置为 0。CRC 模式 DLLDP 示意图见图 4-37。

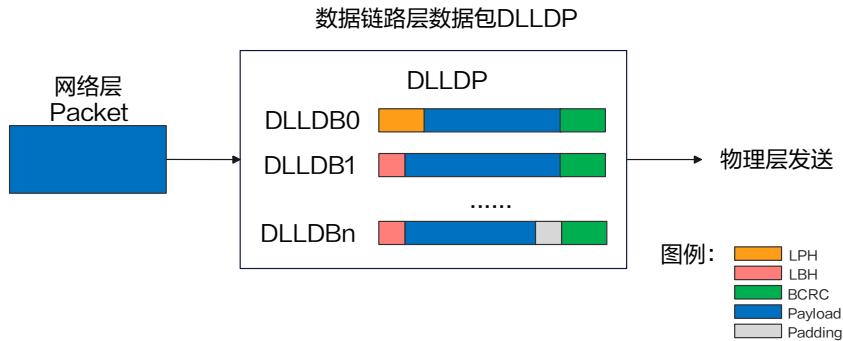


图 4-37 DLLDP 发送流程 ( CRC 模式 )

DLLCB 具有固定的格式，应符合 4.3.3 节相关要求。DLLCB 由数据链路层生成后下发到物理层。

#### 4.3.4.3 接收流程

接收端接收 DLLCB 和 DLLDP 时，若使用 CRC 模式，则在数据链路层判断物理层 FEC 解码状态（启用物理层 FEC 时）和 CRC 校验结果，若 FEC 解码失败或 CRC 校验失败，则触发数据链路层重传流程；若使用 Non-CRC 模式，则在数据链路层判断物理层 FEC 解码状态，若 FEC 解码失败，则触发数据链路层重传流程。

Block 通过上述校验后，根据定界结果以及长度信息，识别头部域段 LCH/LPH/LBH，并进行头部域段解析。

若接收的为 DLLDP，接收端根据 DLLDP 中 PLENGTH 域段，获取 Payload 的边界，丢弃 Padding（如果有），并将 Payload 上送至网络层；若接收的为 DLLCB，则由数据链路层完成后续链路管理控制，无需发送至网络层（Param\_Exchg Block 除外）。

## 4.4 初始化自协商

数据链路在 DLL\_Disabled 状态下，检测到物理层已建链后，进入 DLL\_Param\_Init 状态进行初始化，应符合 4.2 节相关要求。

在 DLL\_Param\_Init 状态下数据链路层通过收发 Init Block 完成数据链路层的握手，从而完成初始化参数交换，保证后续的正常通信。在自协商时，链路两端的数据链路层发送 Init Block 没有规定的先后顺序，进入 DLL\_Param\_Init 状态即可发送。当在链路上产生误码导致 Init Block 被丢弃时，数据链路层通过重传机制实现 Init Block 重传，保证 Init Block 的正确接收。

数据链路层在初始化过程中，通过 Init Block 获取到对端数据链路层支持的能力，协商后两端数据链路层使用相同的工作参数，工作在相同模式。协商规则见表 4-9：

表 4-9 数据链路层初始化自协商规则

域段名称	含义	协商规则（示例见下文）
DATA_CREDIT_GRAIN_SIZE	DLLDP头部LPH/LBH域段中VL0~VL15的信用证返还粒度（单位：Cell）； 每个Byte对应一个VL的信用证返还粒度 {DATA_CREDIT_GRAIN_SIZE_VL15, DATA_CREDIT_GRAIN_SIZE_VL14,……, DATA_CREDIT_GRAIN_SIZE_VL0}； 每bit对应一个Cell数量，分别为 {128,64,32,16,8,4,2,1}。	两端支持的信用证返还粒度取交集，交集中最小数值作为协商后的结果。
CTRL_CREDIT_GRAIN_SIZE	Crd_Ack Block支持VL0~VL15的信用证返还粒度（单位：Cell）； 每个Byte对应一个VL的信用证返还粒度 {CTRL_CREDIT_GRAIN_SIZE_VL15, CTRL_CREDIT_GRAIN_SIZE_VL14,……, CTRL_CREDIT_GRAIN_SIZE_VL0}； 每bit对应一个Cell数量，分别为 {128,64,32,16,8,4,2,1}。	两端支持的信用证返还粒度取交集后，交集中最小数值作为协商后的结果。
FEATURE_ID	数据链路层协议版本号，本规范支持的值为1。	协商后取较低版本的协议版本号，作为两端共同的工作模式。
RXBUF_VL_SHARE	Receive Buffer是否支持VL共享，1'b1：支持；1'b0：不支持。	当对端Receive Buffer不支持VL共享时，本端发送方使用信用证独占模式做信用流控；当对端Receive Buffer支持VL共享时，本端发送方使用信用证独占或共享模式均可。
DATA_ACK_GRAIN_SIZE	LPH/LBH中支持的ACK返还粒度（单位：Flit）； 每bit对应一个Flit数量，分别为 {128,64,32,16,8,4,2,1}。	两端支持的ACK返还粒度取交集后，交集中最小数值作为协商后的结果。
CTRL_ACK_GRAIN_SIZE	Crd_Ack Block支持ACK返还粒度（单位：Flit）；每bit对应一个Flit数量，分别为 {128,64,32,16,8,4,2,1}。	两端支持的ACK返还粒度取交集后，交集中最小数值作为协商后的结果。
FLOW_CTRL_SIZE	支持信用流控最小单元Cell对应的Flit数量； 每bit对应一个Flit数量，分别为 {128,64,32,16,8,4,2,1}。	两端支持的Cell粒度取交集后，交集中最小数值作为协商后的结果。

域段名称	含义	协商规则（示例见下文）
VL_ENABLE	支持启用的最多16个虚拟通道，bit0~bit15分别表示VL0~VL15，VL0必须启用； 1'b1：表示支持使用此VL；1'b0：表示不支持使用此VL。	两端支持的VL从0开始，连续编号，取交集后即为启用的虚拟通道。

在设备出厂时，会将设备的能力记录在 Data Link Capability 寄存器中，具体寄存器定义参考附录 D.6.2.1 节，该寄存器的内容对用户为只读，用户无法修改。在两端进行初始化协商时，将想要协商的工作模式记录在 Data Link Configuration 寄存器中，具体寄存器定义参考附录 D.6.2.2 节。在两端设备自协商完成后，将协商启用的能力记录在 Data Link Status 寄存器中，Data Link Status 寄存器用于记录 Link 实时的状态，具体寄存器定义参考附录 0 节，该寄存器的内容对用户为只读，用户无法修改，由硬件实时修改维护。

为保证初始化协商一定成功，要求数据链路层必须具备如下能力，下列能力硬件默认支持，软件无需配置：

1. Crd\_Ack Block 必须具备信用证返还粒度 1 Cell 能力；
2. DLLDP 必须具备信用证返还粒度 4 Cells 能力；
3. Crd\_Ack Block 必须具备 ACK 返还粒度 1 Flit 能力；
4. DLLDP 必须具备 ACK 返还粒度 32 Flits 能力；
5. 必须具备信用流控最小单元 Cell 按 8 Flits 处理的能力；
6. 必须具备通过 VL0 进行 DLLDP 传输的能力，即 VL\_ENABLE[0]=1'b1。

示例：协商示例

以协商 FLOW\_CTRL\_SIZE 为例：在配置 FLOW\_CTRL\_SIZE 粒度时，8 Flits 默认支持，软件无需配置，软件只需配置希望支持的值。

假设 DL0 配置为{32,16}，DL1 配置为{64,32,16}，则协商后 FLOW\_CTRL\_SIZE 的取值为 16。假设 DL0 配置为{32}，DL1 配置为{16}，则协商失败，采用默认的工作模式，FLOW\_CTRL\_SIZE 的取值为 8。

数据链路层各个域段初始化协商不成功时默认工作模式为：

1. DATA\_CREDIT\_GRAIN\_SIZE=0x040404040404040404040404040404 ( 在 16 个 VL 中，DLLDP 的每个 VL 信用证返还粒度均为 4 Cells )；
2. CTRL\_CREDIT\_GRAIN\_SIZE=0x010101010101010101010101010101 ( 在 16 个 VL 中，Crd\_Ack Block 每个 VL 的信用证返还粒度均为 1 Cell )；
3. FEATURE\_ID=0x0001 ( 本规范该值为 1 )；
4. RXBUF\_VL\_SHARE=0x0 ( Receive Buffer 不支持共享信用证 )；
5. DATA\_ACK\_GRAIN\_SIZE=0x20 ( DLLDP 的 ACK 返还粒度为 32 Flits )；
6. CTRL\_ACK\_GRAIN\_SIZE=0x01 ( Crd\_Ack Block 的 ACK 返还粒度为 1Flit )；

7. FLOW\_CTRL\_SIZE=0x08 ( 1 个信用证大小为 8 Flits, 即 1 Cell=8 Flits ) ;
8. VL\_ENABLE=0x1 ( VL0 启用, 其它 VL 都禁用 ) 。

## 4.5 虚拟通道机制

### 4.5.1 虚拟通道介绍

VL 是一种在单个物理通道上实现多个逻辑通信通道的机制。数据链路层每个点到点链路支持最多 16 个 VL, 信用流控基于 VL 粒度实现。DLLDP 内的所有 DLLDB 使用同一个 VL 传输, 不同的 DLLDP 可能会使用不同的 VL。同一 VL 内的 DLLDP 以先来先服务 (First Come First Service, FCFS) 的方式进行传输, 不同 VL 的传输顺序与实现的调度算法相关。

DLLCB 不占用信用证, 因此传输 DLLCB 时不指定 VL。

### 4.5.2 虚拟通道标识

数据链路层在传输 DLLDP 时, 通过该 DLLDP 的 LPH.VL 和 LBH.VL 来表示其对应的虚拟通道。

## 4.6 信用流控机制

### 4.6.1 信用证初始化

#### 4.6.1.1 信用证介绍

数据链路层支持信用流控, 每个链路的接收端根据 Receive Buffer 的大小和初始化参数分配相应数量的信用证给发送端, 对发送端通过信用证进行流控, 从而防止接收端的缓存溢出导致 DLLDP 丢失。信用流控的基本单元是信用证 Cell, 1 个 Cell 对应 1 个或多个 Flits, 具体对应数量在参数初始化阶段由两端链路协商确定, 具体见 4.4 节。

数据链路层每个链路在开始传输 DLLDP 之前, 通过发送 Crd\_Ack Block 将本端 Receive Buffer 的接收能力告知对端, 此过程在 DLL\_Credit\_Init 状态下完成。

数据链路层每个链路在完成初始化自协商后会启用部分或全部 VL, 其中 VL0 一定会被启用, 其它 VL 根据初始化自协商结果决定是否启用。信用证支持独占和共享两种模式, 同样在初始化自协商时决定是否启用信用证共享模式, 初始化自协商应符合 4.4 节相关要求。

### 4.6.1.2 信用证独占模式

在初始化信用证之前，两端通过 Init Block 完成了初始化操作，此时已协商好需要启用的 VL、Cell 粒度大小和信用证独占模式。本端根据 Receive Buffer 大小和 Cell 粒度计算支持的信用证数量，根据管理面配置的策略为启用的每个 VL 分配独占的信用证，即各个 VL 使用各自的信用证，互不干扰。随后链路两端互相将本端初始化的信用证通过 Crd\_Ack Block 发送给对端完成初始化操作。

示例：独占信用证工作流程

假设本端 Receive Buffer 空间为 1MB，自协商结果如下：

1. 初始化自协商后启用了 VL0~VL8 共 9 个 VL；
2. 初始化自协商后 1 Cell=8 Flits；
3. 本端 Receive Buffer 空间不支持 VL 共享，即每个 VL 独立占用一块 Receive Buffer 空间。

基于上述自协商结果，初始化信用证总数应为  $1\text{MB}/(8*20\text{B})=6553 \text{ Cells}$  ( 计算结果向下取整，保证 Receive Buffer 不会溢出 )，用户根据实际需要将 6553 Cells 信用证分配给 9 个 VL 使用，记每个 VL 的信用初值为 INIT\_CRD0~INIT\_CRD8，原则上要求 INIT\_CRD0+INIT\_CRD1+……+INIT\_CRD8 的总数不超过 6553 Cells，具体的分配策略可根据实际场景需要设计。

返还信用证时，信用证返还到对应的 VL 上，各个 VL 的信用证互相独立。

### 4.6.1.3 信用证共享模式

在初始化自协商时，链路两端协商使用信用证共享模式。此模式下，初始化流程与信用证独占模式下的流程类似，不同之处在于信用证共享模式下，会将所有接收到的信用证记录到共享信用证计数器 SHARE\_CRD 中。

为避免 VL 长期饥饿得不到信用证分配，信用证共享模式下也需要给 VL 分配独占信用证，即根据管理面配置的策略从 SHARE\_CRD 中分配部分信用证作为独占信用证给启用的 VL。独占信用证不会被其它 VL 占用，独占数量根据实际场景定义。

信用证初始化完成后，接收到对端返还的信用证时，将返还的信用证汇总到 SHARE\_CRD 中，并检查各个 VL 独占信用证是否充足，即是否与初始化时分配的信用证数量一致，不足则进行补充，补充顺序可根据实际场景定义，例如设置各个 VL 补充信用证优先级。

对比各个 VL 独占部分信用证的模式，VL 间共享信用证的信用流控方案会提高 Receive Buffer 的资源利用率，发送端可以根据各个 VL 的流量状态等信息动态调整各个 VL 信用证数量以提高传输性能。

示例：共享信用证工作流程

假设本端 Receive Buffer 空间为 1MB，自协商和管理面配置结果如下：

1. 初始化自协商后启用 VL0、VL1 两个 VL；
2. 初始化自协商后 1 Cell=8 Flits；
3. 本端与对端协商启用共享信用证模式；
4. 对端设置 VL0、VL1 的独占信用证值分别为 128 Cells、128 Cells，数量可根据实际场景设计。

基于上述结果，初始化信用证总数应为  $1\text{MB}/(8*20\text{B})=6553\text{ Cells}$ ，对端将所有信用证汇总到共享信用证计数器 SHARE\_CRD 中，并分配 VL0、VL1 独占的信用证。完成初始的信用证分配后，各计数器的值如下：

1. VL0 独占信用证计数器值为 128；
2. VL1 独占信用证计数器值为 128；
3. SHARE\_CRD 值为 6297。

本示例中，发送 DLLDP 时，各 VL 优先使用共享信用证发送，当共享信用证数量不足时，使用本 VL 独占的信用证发送，无需等待共享信用证返还。

假设当前共享信用证剩余 64 Cells，VL1 待发送 DLLDP 需消耗 128 Cells，共享信用证不足，需使用 VL1 独占的 64 Cells 信用证凑齐 128 Cells。

返还信用证时，将信用证汇总到 SHARE\_CRD 中，并检查各个 VL 独占信用证是否充足，即信用证计数器值是否为 128，若不足，则补充各 VL 独占的信用证，补充顺序可根据实际场景定义。

## 4.6.2 信用流控过程

数据链路层信用流控过程见图 4-38：

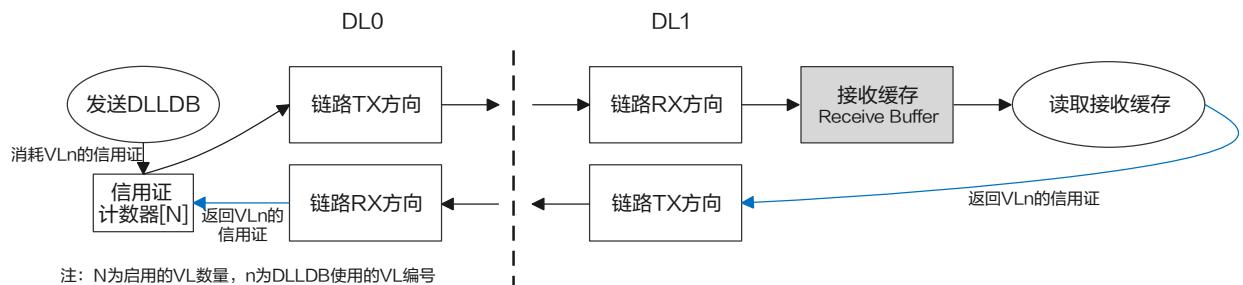


图 4-38 信用流控过程

数据链路层的信用流控过程如下（此处使用信用证独占模式）：

1. DL1 在 DLL\_Credit\_Init 状态时通过链路 TX 方向发送 Crd\_Ack Block，为 DL0 的各个 VL 分配初始信用证，信用证总数量对应 Receive Buffer 的空间大小；
2. DL0 基于每个 VL 维护信用证计数器，信用证计数器用于记录对端 Receive Buffer 中目前可用的 Buffer 空间大小。RX 接收到返还信用证的 DLLCB/DLLDP，信用证计数器则增加对应数量的信用证；TX 发送 DLLDP，信用证计数器则减去 DLLDP 占用的信用证数量。DL0 的 VL 持有足够的信用证才能将 DLLDP 发送到 DL1 对应 VL 的 Receive Buffer 中；
3. DL1 Receive Buffer 中的 DLLDP 被读走后释放对应的信用证，释放的信用证数量嵌入到发往 DL0 的 DLLDP 或 Crd\_Ack Block 中返还给 DL0 对应的 VL；
4. DL0 收到释放的信用证后，将其加到对应 VL 的信用证计数器中。

注 1：信用证的基本单元为 Cell=n\*Flit，n 的取值由 Init Block 中的 FLOW\_CTRL\_SIZE 域段协商确定，n 的取值范围为 {1,2,4,8,16,32,64,128}；

注 2：信用流控支持的最大信用证数量为 65535 Cells，该数量指可分配给所有已启用 VL 的总信用证量，实际使用的信用证数量根据硬件能力和协商结果确定；

注 3：DLLDP 消耗的信用证数量计算方法：DLLDP 内的 Flit 数量除以 1Cell 表示的 Flit 数量并向上取整。为避免信用证数量异常，消耗和返还的信用证计算方式需保持一致。

注 4：通过 LPH/LBH 中的 CRD 和 CRD\_VL 域段返还信用证到发送端对应的 VL 时，CRD 的返还粒度为  $m^*Cell$ ， $m$  的取值由 Init Block 中的 DATA\_CREDIT\_GRAIN\_SIZE 域段协商确定， $m$  的取值可以为{1,2,4,8,16,32,64,128}；

注 5：通过 Crd\_Ack Block 的 CRD\_NUM[95:0]域段返还信用证到发送端对应的 VL 时，其中 CRD 的返还粒度为  $k^*Cell$ ， $k$  的取值由 Init Block 中的 CTRL\_CREDIT\_GRAIN\_SIZE 域段协商确定， $k$  的取值可以为{1,2,4,8,16,32,64,128}。

发送端通过信用流控机制实时监测接收端 Receive Buffer 的剩余可用空间，在原理上保证接收端 Receive Buffer 不会溢出，接收端无需考虑链路延时等因素提前反压发送端。

### 4.6.3 信用证返还规则

信用证可以通过 DLLDP 的 LPH/LBH 域段和 Crd\_Ack Block 来返还到对端，信用证独占模式与信用证共享模式下返还规则一致。信用证返还规则如下：

1. 本端维护计数器记录待返还信用证的数量，当数量达到返还粒度时，若本端有 DLLDP 要发送，则通过 DLLDP 的 LPH/LBH 域段将信用证返还到对端。若一个 DLLDP 内有多个 LPH/LBH 的 CRD 域段置为 1，则返还多次信用证。
2. 本端无 DLLDP 发送时，若仍有信用证要返还，则通过发送 Crd\_Ack Block 将信用证返还到对端。
3. 当本端待返还的信用证达到阈值时（每个 VL 单独设置阈值，阈值大小根据实际应用配置），需要反压 DLLDP 发送通道，即反压网络层，并强制发送 Crd\_Ack Block，从而及时返还信用证到对端。避免本端发送方向通过 DLLDP 的 LPH/LBH 域段无法返还足够的信用证给对端时，影响对端发送 DLLDP 的性能。同时，为减少发送端发送 Crd\_Ack Block 对发送性能的影响，初始化协商时应尽量保证配置的 LPH/LBH 返信用证粒度能够满足带宽线速要求。

## 4.7 误码检测和重传机制

### 4.7.1 重传触发方式

数据链路层在 Non-CRC 模式、CRC 模式下均支持重传，保证 DLLCB/DLLDP 在链路上传输的可靠性。

在 CRC 模式下，数据链路层在发送端为 Block 添加 BCRC 域段，BCRC 中包含 Block 的 CRC 校验信息，发送端发出 Block 时在 Retry Buffer 进行备份（Null Block、Retry Block 除外）。接收端接收 Block 时，判断物理层 FEC 解码状态（启用物理层 FEC 时），若 FEC 解码成功则进行 CRC 校验，若 FEC 解码失败或 CRC 校验失败则发出重传请求，触发发送端数据链路层进行重传，CRC 校验流程见 4.7.2 节。

在 Non-CRC 模式下，接收端数据链路层接收 Block 时，判断 Block 对应的物理层 FEC 解码状态，若 FEC 解码失败则发出重传请求，触发发送端数据链路层进行重传。

### 4.7.2 CRC 校验

在 CRC 模式下，发送端计算 DLLDB 的 CRC，CRC 计算的结果携带在 BCRC 域段中。接收端计算接收到的 DLLDB 的 CRC，并与 Block 中携带的 BCRC.CRC30 域段进行对比较验。

BCRC 由 2 bits 信息位 ( bit[31]: Reserved, bit[30]: ERROR\_FLAG, 用于标错 ) + 30 bits 的 CRC30 构成，CRC 多项式为： $x^{30} + x^{28} + x^{26} + x^{24} + x^{23} + x^{21} + x^{19} + x^{16} + x^{14} + x^{11} + x^9 + x^7 + x^6 + x^4 + x^2 + 1$ 。

其中：

1. CRC 计算的初值为全 1；
2. 根据 DLLDB 计算 CRC 时，按照 Byte0-Bit7、Byte0-Bit6 的顺序进行计算，具体的 CRC 计算位序见图 4-39；
3. CRC 计算结果不需要取反；
4. CRC 计算结果和 CRC30 域段的 bit 位序一一对应，无需调整位序；
5. CRC30 校验覆盖本 DLLDB 中 CRC30 域段前的所有数据。

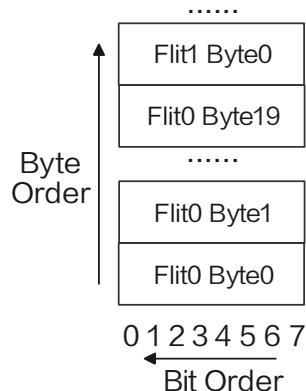


图 4-39 CRC 计算位序

### 4.7.3 重传机制

#### 4.7.3.1 重传机制概述

数据链路层在发送端将已发出的 Block 备份到 Retry Buffer（相关内容见 4.7.3.2 节）中，在接收端 CRC 校验出错或 FEC 解码失败时进行重传，数据链路层采用 GoBackN 重传机制，即重传 CRC 校验出错的 Block 或 FEC 解码失败的帧对应的首个 Flit，以及 Retry Buffer 中该 Flit 之后的所有 Flits。

接收端在数据链路层 DLL\_Param\_Init 状态下通过 Init Block 获取发送端 Retry Buffer 深度 RETRY\_BUF\_DEPTH( 相关内容见 4.3.3.9 节 ), 在此基础上, 计算并维护 RcvPtr( 相关内容见 4.7.3.2.2 节 ), RcvPtr 用于指示本端等待接收的 Flit 在对端 Retry Buffer 中的位置, 每个 Flit 对应一个唯一确定的编号。

在整个重传处理流程中, 重传由接收端触发, 接收端的处理流程称为收端重传流程, 由重传请求状态机 ( 相关内容见 4.7.3.3 节 ) 控制。发送端响应重传, 重新发送 Retry Buffer 中备份的待重传的 Flits, 发送端的处理流程称为发端重传流程, 由重传应答状态机 ( 相关内容见 4.7.3.4 节 ) 控制。

### 4.7.3.2 Retry Buffer 管理

#### 4.7.3.2.1 Retry Buffer 编号与存放内容

Retry Buffer 为发送端缓存, 用于存储已发送但未被确认应答的 Flits。Retry Buffer 的最低地址编号为 0, 最高地址编号为 RETRY\_BUF\_DEPTH-1, 空间中任一位置地址表示为相对于 0 的偏移量, 以 Flit 为单位。当发送端接收到重传请求时, 发送端会根据重传请求中的信息, 将 Retry Buffer 中对应的备份 Flits 重新发送, 以实现数据链路层的可靠传输功能。

在 Retry Buffer 中备份的 Flits 来自 DLLDP 以及 DLLCB ( Null Block、Retry Block 除外 )。

#### 4.7.3.2.2 Retry Buffer 空间管理参数

数据链路层通过如下控制参数 ( 单位均为 Flit ) 来进行 Retry Buffer 的空间管理:

1. NumFreeBuf: 表示 Retry Buffer 的空闲空间。当本端成功接收到 ACK 后 ( 包括 DLLDP 通过 LPH/LBH 的 ACK 域段返回、Crd\_Ack Block 通过 ACK\_NUM 域段返回两种 ACK 返回方式 ), 根据 ACK 类型和对应的粒度计算出本端 Retry Buffer 空间需要释放的 Flit 数量, 记为 ReleaseSize, 若  $\text{NumFreeBuf} + \text{ReleaseSize} \leq \text{RETRY\_BUF\_DEPTH}$ , 则  $\text{NumFreeBuf} = \text{NumFreeBuf} + \text{ReleaseSize}$ , 否则触发 ACK 溢出异常处理, 溢出异常处理相关见 4.8.2 节。当有长度为 SendSize 个 Flits 的 Block 尝试发送时, 若  $\text{NumFreeBuf} \geq \text{SendSize}$ , 才允许发送, 并更新  $\text{NumFreeBuf} = \text{NumFreeBuf} - \text{SendSize}$ 。本端成功接收到 ACK 或备份 Block 到 Retry Buffer 时, 首先校验 NumFreeBuf, 未发生溢出异常和满足发送条件时, 才会执行以下的 WrPtr 和 TailPtr 参数的改动;
2. WrPtr: 表示最新发送的 Flit 备份到 Retry Buffer 的位置。本端每发送一个需要备份的 Flit, 就将该 Flit 写入 Retry Buffer 的 WrPtr 地址处, 若  $\text{WrPtr} + 1 == \text{RETRY\_BUF\_DEPTH}$ , 则 WrPtr 归零, 否则  $\text{WrPtr} = \text{WrPtr} + 1$ , 然后继续备份后续发出的 Flit;
3. TailPtr: 表示 Retry Buffer 中所有待返回 ACK 的 Flit 的起始位置。本端接收到 ACK 后, 根据 ACK 类型和对应的粒度计算出本端需要释放的 Buffer 空间大小, 记为 ReleaseSize, 若  $\text{TailPtr} + \text{ReleaseSize} \geq \text{RETRY\_BUF\_DEPTH}$ , 则  $\text{TailPtr} = \text{TailPtr} + \text{ReleaseSize} - \text{RETRY\_BUF\_DEPTH}$ , 否则  $\text{TailPtr} = \text{TailPtr} + \text{ReleaseSize}$ ;

4. RdPtr：表示下一个待重传的 Flit 在 RetryBuffer 的位置。当收到重传请求时，RdPtr 被置为重传请求中携带的 RcvPtr，之后每读取发出一个 Flit，若  $RdPtr + 1 == RETRY\_BUF\_DEPTH$ ，则  $RdPtr = 0$ ，否则  $RdPtr = RdPtr + 1$ ，直至  $RdPtr == WrPtr$ 。RdPtr == WrPtr 的位置无数据，该位置不需要重传；
5. RcvPtr：表示 Block 接收端等待接收的 Flit 在对端 Retry Buffer 中的位置。当重传请求状态机（RETRY\_REQ\_SM，见 4.7.3.3 节）在 NORMAL 状态下，若使用 CRC 模式，每正确接收到 Flit 数量为 ReceivedSize 的 DLLDB，RcvPtr 增加 ReceivedSize；若使用 Non-CRC 模式，每正确接收到一个 Flit，RcvPtr 增加 1，此时  $ReceivedSize == 1$ 。若  $RcvPtr + ReceivedSize >= RETRY\_BUF\_DEPTH$ ，则  $RcvPtr = RcvPtr + ReceivedSize - RETRY\_BUF\_DEPTH$ ，否则  $RcvPtr = RcvPtr + ReceivedSize$ ；当发生 CRC 校验错误或 FEC 解码失败时，Block 接收端通过携带 RcvPtr 的 Retry\_Req Block 通知对端从 RcvPtr 处开始，按照 GoBackN 机制进行重传。

RcvPtr 由重传发起端（数据接收端）维护，WrPtr、TailPtr、RdPtr、NumFreeBuf 由重传响应端维护。链路状态机进入到 DLL\_Disabled 状态时，Retry Buffer 空间管理参数会进行初始化，RcvPtr、WrPtr、TailPtr、RdPtr 初始值均为 0，NumFreeBuf 初始值为 RETRY\_BUF\_DEPTH。

Retry Buffer 空间管理示意图如图 4-40 所示。每个编号标识 1 个 Flit 大小的空间，RETRY\_BUF\_DEPTH 为 N，WrPtr 为 N-2，TailPtr 为 2，白色部分表示 Retry Buffer 空闲空间，其大小为 NumFreeBuf，灰色部分表示 Retry Buffer 已被占用的空间。

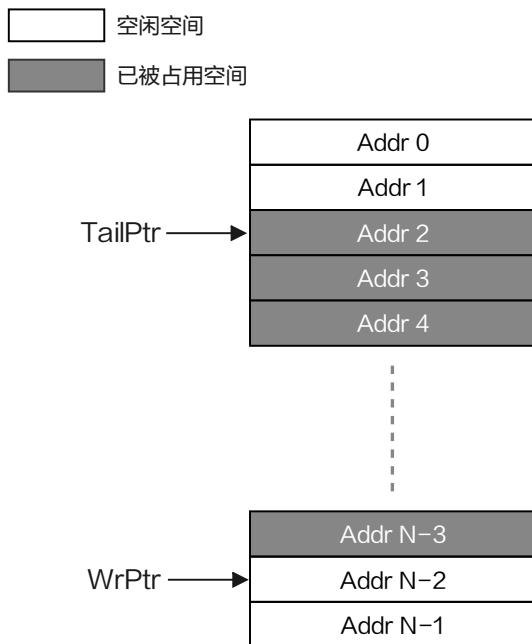


图 4-40 Retry Buffer 空间管理示意图

#### 4.7.3.2.3 Retry Buffer 空间释放

在 CRC 模式下, 当 DLLDP 中 DLLDB 的物理层 FEC 解码成功(启用物理层 FEC 时), 且 CRC 校验通过被本端正确接收后, 或在 Non-CRC 模式下, 当 DLLDP 中 Flit 的物理层 FEC 解码成功被本端正确接收后, 可通过以下两种方式告知对端释放对应的 Retry Buffer 空间:

1. 基于 DLLDP 的随路 ACK 机制: 即通过置位本端发送方向的 DLLDP 的 LPH.ACK 域段, 或置位 DLLDP 中 DLLDB 的 LBH.ACK 域段, 通知对端释放数量为 DATA\_ACK\_GRAIN\_SIZE 个 Flits 的 Retry Buffer 空间;
2. 基于 Crd\_Ack Block 的 ACK 机制: 即发送 Crd\_Ack Block, 通过其 ACK\_NUM 域段通知对端释放 ACK\_NUM\*CTRL\_ACK\_GRAIN\_SIZE 个 Flits 的 Retry Buffer 空间。

#### 4.7.3.2.4 Retry Buffer 预防互锁

由于携带 ACK 的 Block 在发送时都需要在发送端 Retry Buffer 中进行备份, 而发送端 Retry Buffer 空间的释放要依赖于接收端接收携带 ACK 的 Block, 所以存在 Retry Buffer 在收发两端形成互锁的可能, 例如:

数据链路层在传输过程中由于误码率高, 导致链路两端均无法快速返还 ACK 给对端释放 Retry Buffer, 因此两端 Retry Buffer 被写满后无法发送新的 Block, 而此时待返还的 ACK 全部积攒到本地无法发出, 且 Retry Buffer 中没有备份携带 ACK 返还信息的 Block。此时链路两端在数据链路层形成互锁, 无法继续通信。

为预防上述情况发生, 对 Retry Buffer 定义 Crd\_Ack Block 优先发送阈值从而避免链路两端通信互锁。当 Retry Buffer 的 NumFreeBuf 小于此阈值后, 暂停 DLLDP 和其余 DLLCB 的发送, 优先发送 Crd\_Ack Block 通知对端释放 Retry Buffer, 以保证在 Retry Buffer 中一定备份了携带 ACK 的 Crd\_Ack Block, 从而解除互锁问题。

#### 4.7.3.3 重传请求状态机 (RETRY\_REQ\_SM)

当接收到的 CRC 模式 Block 的物理层 FEC 解码失败或 CRC 校验失败, 或 Non-CRC 模式 Flit 的物理层 FEC 解码失败后, Block 接收端发起数据链路层重传请求, 通过重传请求状态机管理本端重传流程, 重传请求状态机见图 4-41。

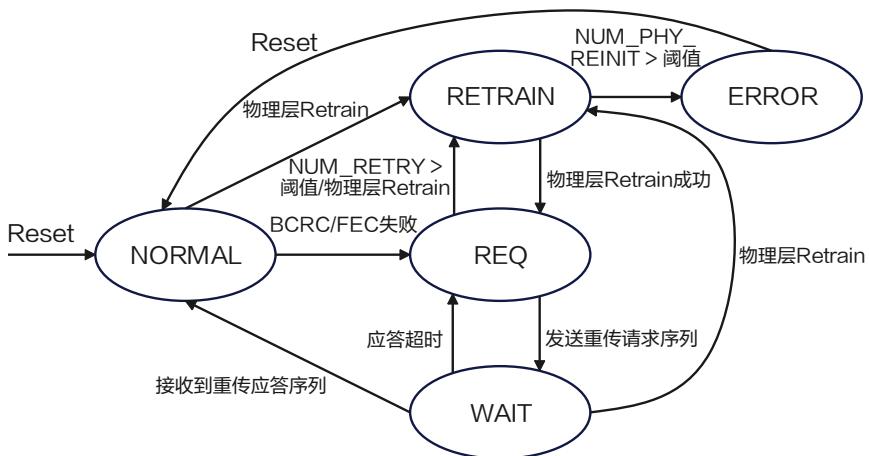


图 4-41 重传请求状态机

重传请求状态机包含 5 个状态，定义如下：

1. NORMAL：正常状态，此状态下正常收发 Block。当出现 FEC 解码失败或 CRC 校验失败或物理层 Retrain 时（物理层会为数据链路层提供物理层 Retrain 标志位，数据链路层根据该标志位进入 RETRAIN 状态，相关内容见 3.4.3 节），触发状态迁移。
  - (1) 在此状态设置计数器 NUM\_RETRY 的值为 0，用于记录同一重传事件内物理层 Retain 之前发起重传请求的次数，物理层 Retain 后清 0，重新计数；设置计数器 NUM\_PHY\_REINIT 的值为 0，用于记录物理层 Retain 次数。
2. REQ：发送重传请求序列状态，此状态下判断是否满足进入 RETRAIN 的条件，如不满足则向对端发送重传请求序列。规定如下：
  - (1) 为保证对端正确识别出重传请求，必须连续发送 1 个 Retry\_Idle Block 和 32 个 Retry\_Req Block，即发送一个重传请求序列 Retry\_Req\_Set，否则可能导致 DLLDP 与重传请求识别混乱；
  - (2) 此状态下数据链路层只支持接收 Retry\_Idle Block、Retry\_Req Block 和 Retry\_Ack Block 三种 DLLCB，其它类型的 DLLCB/DLLDP 均会丢弃；
  - (3) 每次进入 REQ 状态，NUM\_RETRY 的值加 1，当 NUM\_RETRY 值等于 NUM\_RETRY\_THRESHOLD 或物理层 Retain 时，触发状态迁移至 RETRAIN，NUM\_RETRY 值清 0。
3. WAIT：等待重传应答状态，此状态下等待接收对端发送的重传应答序列。规定如下：
  - (1) 此状态下数据链路层只支持接收 Retry\_Idle Block、Retry\_Req Block 和 Retry\_Ack Block 三种 DLLCB，其它类型的 DLLCB/DLLDP 均会丢弃；
  - (2) 此状态下设置超时计数器，计数器的值超过阈值后触发状态迁移至 REQ。超时计数器阈值对应的时间建议设置为大于 2 倍链路时延，推荐时间范围为 1μs 到 10s。
4. RETRAIN：此状态下判断是否满足进入 ERROR 状态的条件，如不满足则向物理层发送 Retain 命令，并等待物理层 Retain 成功。若物理层长时间不响应 Retain 命令，物理层将上报 LinkUp==1'b0，数据链路层恢复到 DLL\_Disabled 状态。
  - (1) 每次进入 RETRAIN 状态，NUM\_PHY\_REINIT 计数器值加 1，当 NUM\_PHY\_REINIT 值等于 NUM\_PHY\_REINIT\_THRESHOLD 时，触发状态迁移至 ERROR，NUM\_PHY\_REINIT 值清 0；
  - (2) 此状态下数据链路层不支持 Block 收发。
5. ERROR：此状态下数据链路层不支持 Block 收发，需上报错误，等待 UBFM ( UB Fabric Manager ) 复位。复位后，重传请求状态机重新进入 NORMAL 状态。

示例：阈值设置示例

NUM\_RETRY\_THRESHOLD 建议阈值为 15，NUM\_PHY\_REINIT\_THRESHOLD 建议阈值为 4，可根据实际场景灵活定义。

每个状态转移规则如下：

1. 进入 NORMAL 状态条件：
  - (1) 链路复位；
  - (2) 在 WAIT 状态下，收到了对端发送的重传应答序列。
2. 进入 REQ 状态条件：
  - (1) 在 NORMAL 状态下，检测到 Flit 对应的物理层 FEC 解码失败或 Block 的 CRC 校验失败；
  - (2) 在 WAIT 状态下，等待重传应答序列超时；
  - (3) 在 RETRAIN 状态下，物理层 Retrain 成功。
3. 进入 WAIT 状态条件：
 

在 REQ 状态下，发送重传请求序列完成。
4. 进入 RETRAIN 状态条件：
  - (1) 在 NORMAL 状态下，物理层 Retrain；
  - (2) 在 REQ 状态下，重传次数计数器  $\text{NUM\_RETRY} == \text{NUM\_RETRY\_THRESHOLD}$ ，或物理层 Retrain；
  - (3) 在 WAIT 状态下，物理层 Retrain。
5. 进入 ERROR 状态条件：
 

在 RETRAIN 状态下，RETRAIN 次数计数器  $\text{NUM\_PHY\_REINIT} == \text{NUM\_PHY\_REINIT\_THRESHOLD}$ 。

用户可通过寄存器 Data Link State Machine Status 中的 Retry Req Status 域段获取重传请求状态机的状态，相关内容见 0 节。

#### 4.7.3.4 重传应答状态机 (RETRY\_ACK\_SM)

当接收到重传请求后，进入发端重传流程，由重传应答状态机管理，实现重传应答序列和重传 Block 的发送，状态机见图 4-42。

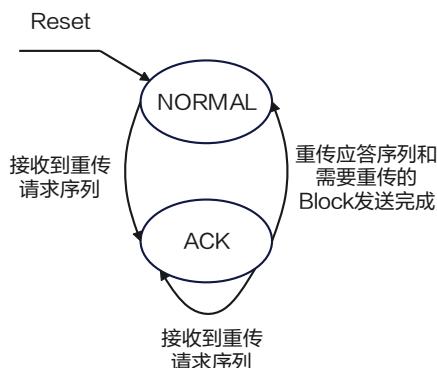


图 4-42 重传应答状态机

重传应答状态机包含 2 个状态：

1. NORMAL：正常状态，此状态下 Block 正常收发，当接收到重传请求序列时，触发状态发生迁移。
2. ACK：发送重传应答序列及重传 Block 状态，此状态下仅可发送重传应答序列和需要重传的 Block，接收 Block 正常进行。为保证对端正确识别出重传应答，必须连续发送 1 个 Retry\_Idle Block 和 32 个 Retry\_Ack Block，即发送一个重传响应序列 Retry\_Ack\_Set，否则可能导致 DLLDP 与重传应答识别混乱。

状态迁移规则如下：

1. 进入 NORMAL 状态条件：
  - (1) 链路复位；
  - (2) 在 ACK 状态下，Retry\_Ack\_Set 和需要重传的 Block 发送完成。
2. 进入 ACK 状态条件：
  - (1) 在 NORMAL 状态下，接收到 Retry\_Req\_Set；
  - (2) 在 ACK 状态下，发送重传 Block 时，接收到对端发送的 Retry\_Req\_Set。

用户可通过寄存器 Data Link State Machine Status 中的 Retry Ack Status 域段获取重传应答状态机的状态，相关内容见 0 节。

#### 4.7.3.5 重传流程

完整的重传流程由 Block 发送端的 RETRY\_ACK\_SM 和 Block 接收端的 RETRY\_REQ\_SM 共同完成。

图 4-43 和图 4-44 分别为正常重传流程和 Retry\_Req\_Set 超时重传流程的示例。示例中使用 CRC 模式。

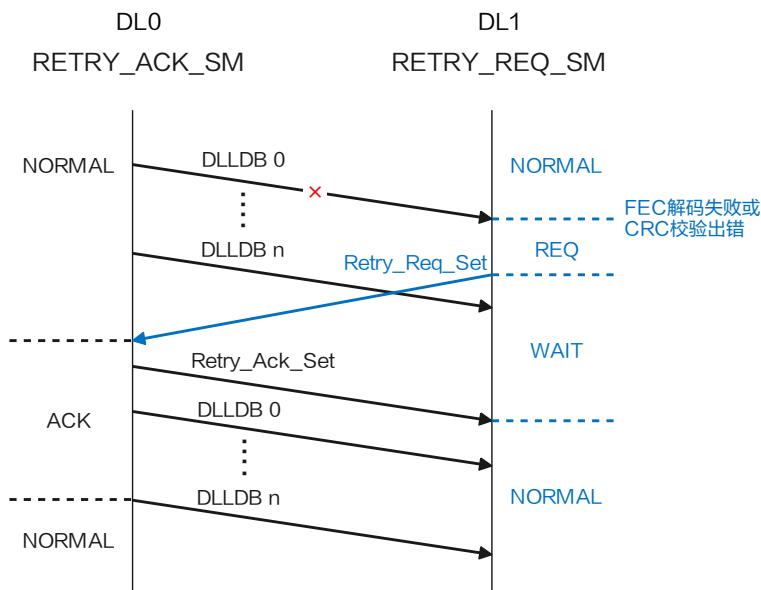


图 4-43 正常重传处理流程示意图

图 4-43 为正常重传流程：

1. DL0 向 DL1 依次发送 DLLDB0~DLLDBn, RETRY\_ACK\_SM 处于 NORMAL 状态;
2. DLLDB0 到达 DL1 后, DL1 对 DLLDB0 的物理层 FEC 解码失败 (启用物理层 FEC 时) 或 CRC 校验失败, RETRY\_REQ\_SM 由 NORMAL 状态迁移到 REQ 状态, DLLDB0 以及后续接收到的 DLLDB1~DLLDBn 均丢弃;
3. DL1 发送 Retry\_Req\_Set, 进入 WAIT 状态;
4. DL0 接收到 Retry\_Req\_Set 后, 进入 ACK 状态, 发送 Retry\_Ack\_Set, 并重传 DLLDB0~DLLDBn, 然后进入 NORMAL 状态;
5. DL1 接收到 Retry\_Ack\_Set 后, 回到 NORMAL 状态。

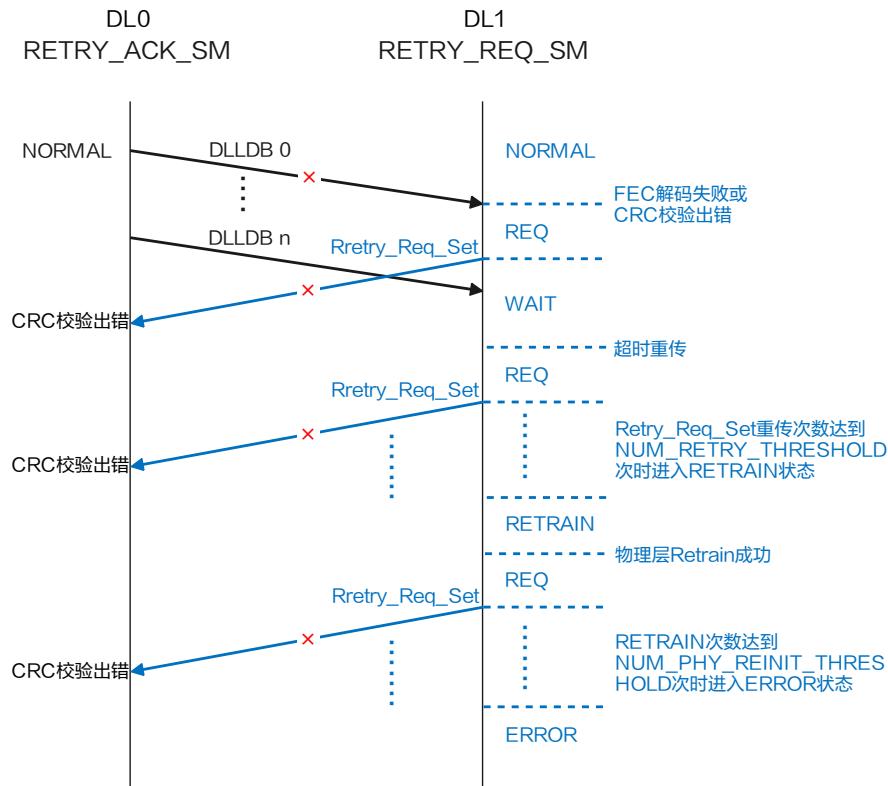


图 4-44 Retry\_Req\_Set 超时重传流程示意图

图 4-44 为 Retry\_Req\_Set 超时重传流程：

1. DL1 发送 Retry\_Req\_Set 进入 WAIT 状态 (前序流程与图 4-43 相同), 重传超时计数器开始计数;
2. DL0 由于误码等原因未能正确接收到 Retry\_Req\_Set, 仍在 NORMAL 状态中继续发送 DLLCB/DLLDP;
3. DL1 在重传超时计数器到达阈值后仍未收到 Retry\_Ack\_Set, 由 WAIT 状态进入 REQ 状态, 重新发送 Retry\_Req\_Set;
4. DL1 连续发送 NUM\_RETRY\_THRESHOLD 次 Retry\_Req\_Set 仍未收到 Retry\_Ack\_Set, 进入 RETRAIN 状态;

5. DL1 在经历 NUM\_PHY\_REINIT\_THRESHOLD 次 RETRAIN 状态后, 依旧未收到 Retry\_Ack\_Set, 进入 ERROR 状态。

## 4.8 异常处理

### 4.8.1 信用证相关异常处理

信用证机制运行过程中, 若发生以下异常, 应上报错误:

1. 若数据链路层接收到的 DLLDB 导致接收端 Receive Buffer 溢出, 应上报 Receive Buffer Overflow 错误;
2. 若数据链路层接收到返还的信用证导致信用证计数超出初始化的信用证数量, 即信用证溢出, 应上报 Flow Control Overflow 错误;
3. 数据链路层每个链路维护信用证返回超时计数器, 若长时间未返回信用证, 则触发信用证返还超时, 应上报 DL Protocol Error 错误。

当出现上述异常时, 数据链路层已无法正常工作, 上报错误后, 等待 UBFM 复位重启, 故障上报和处理流程应符合 10.6.2 节相关要求。

### 4.8.2 重传相关异常处理

重传机制运行过程中, 若发生以下异常, 应上报错误:

1. 若数据链路层发起 Retry\_Req\_Set 后长时间未收到 Retry\_Ack\_Set, 即发送端等待 Retry\_Ack\_Set 超时, 应上报 DL Retry ACK Timeout 错误;
2. 若数据链路层对相同的 RcvPtr 重传 NUM\_RETRY\_THRESHOLD 次后仍然重传失败, 即重传超次, 应上报 DL Retry Rollover 错误;
3. 若数据链路层尝试 NUM\_RETRY\_THRESHOLD\*NUM\_PHY\_REINIT\_THRESHOLD 次重传后仍然重传失败, 重传请求状态机进入 ERROR 状态, 应上报 DL Retry Error 错误;
4. 若数据链路层接收到 ACK 导致 Retry Buffer 的 NumFreeBuf 参数溢出, 应上报 DL Protocol Error 错误。

异常 1 和 2 发生后, 数据链路层自行重试, 具体见 4.7 节。

异常 3 和 4 发生后, 数据链路层已无法正常工作, 上报错误后, 等待 UBFM 复位重启, 故障上报和处理流程应符合 10.6.2 节相关要求。

### 4.8.3 错误 DLLDP 处理

数据链路层在发送 DLLDP 时可以不实现攒齐从而降低发送/转发的时延，即不等待 DLLDP 数据完全就绪，以 DLLDB 粒度发出。可能导致如下错误：当 DLLDP 的前几个 DLLDB 已经发送给物理层后，数据链路层发现读取到的数据有错误（例如内存 ECC 错误），此时在发送方向已经无法丢弃此 DLLDP。

为解决上述问题，数据链路层支持在 DLLDP 的最后一个 Flit 的 ERROR\_FLAG 域段上对此 Packet 进行标错，对端数据链路层将此 Packet 作为正常 DLLDP 解析并发送给上层，最终在 DLLDP 做攒齐的位置将其丢弃。

实现注意事项如下：

1. 数据链路层在接收标错 Packet 时，按正常 DLLDP 来处理，不触发重传（除非有误码触发 CRC 校验错误）；
2. 链路上由误码引入的错误依靠物理层 FEC/数据链路层 CRC 和重传来实现数据恢复，不会将链路误码问题引入到上层。

### 4.8.4 断链处理

物理层检测到断链，上报 LinkUp==1'b0 到数据链路层，数据链路层收到物理层未建链信息后，进行如下处理：

1. 发送端将从上层接收的数据全部丢弃；
2. 若接收端未实现 DLLDP 攒齐，则需要将已部分转发给上层的 DLLDP 补齐（Payload 填 0），并置位 BCRC.ERROR\_FLAG 或 END.ERROR\_FLAG 域段；
3. 将信用计数器清零（应符合 4.6.2 节相关要求）；
4. 将 Retry Buffer 的 WrPtr、TailPtr、RdPtr 和 RcvPtr 清零，NumFreeBuf 置为重传 buffer 的深度值（应符合 4.7.3.2.2 节相关要求）；
5. 数据链路状态机进入 DLL\_Disabled 状态，等待重新初始化。

# 5 网络层

## 5.1 概述

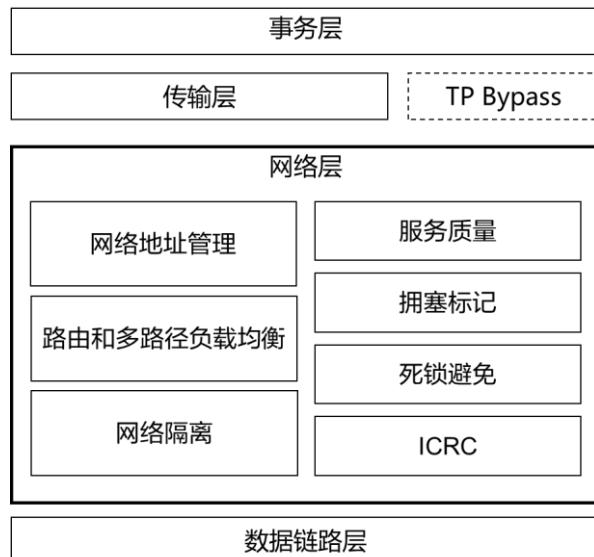


图 5-1 网络层概览

如上图所示，网络层位于数据链路层之上，为传输层和事务层提供服务，主要特性如下：

1. 网络地址管理：网络地址支持标准 IP 地址格式和简短网络地址格式，满足上层协议对兼容性和效率的不同要求，支持动态管理或静态配置方式管理 UB Domain 内的网络地址。
2. 路由和多路径负载均衡：为上层协议提供 UB Domain 内和跨 UB Domain 的路由服务，支持端到端的多路径通信能力，提供逐包和逐流两种负载均衡模式，支持由上层协议指定业务流的工作模式，充分提升多路径网络的利用率。
3. 服务质量：基于 SL-VL 映射及 VL 间调度为上层协议提供差分服务。
4. 拥塞标记：支持多种拥塞标记模式。
5. 网络隔离：支持 IP 网络接口设备间的通信隔离功能。
6. 死锁避免：支持基于路由、VL 主动切换、超时丢包等机制的组合实现死锁避免和解除。
7. ICRC：覆盖数据包中不可被改变的域段，保障端到端数据的完整性。

## 5.2 网络层包头 (NTH)

### 5.2.1 基于 16-bit CNA 地址格式的包头

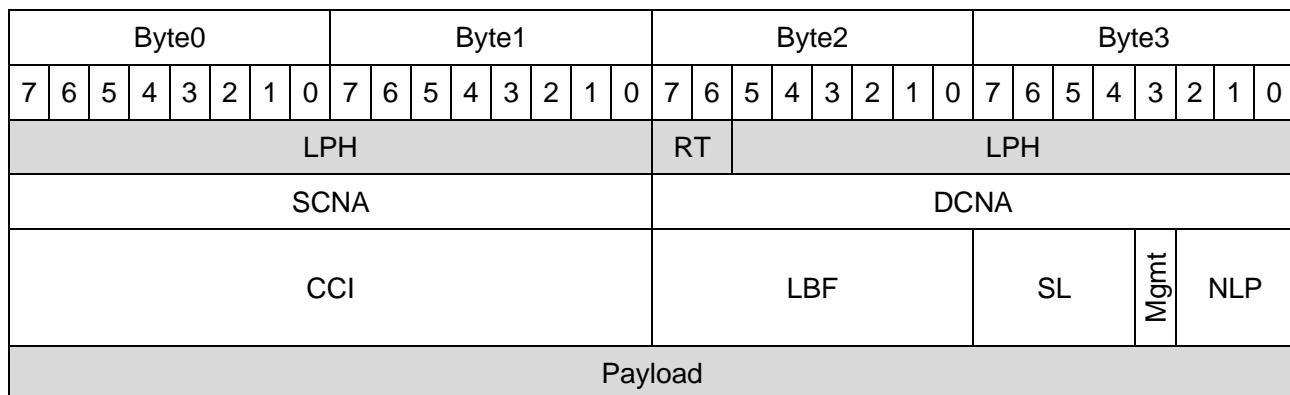


图 5-2 基于 16-bit CNA 地址格式的网络层包头

表 5-1 基于 16-bit CNA 地址格式的网络层包头域段

域段名	位宽(bit)	描述
RT	2	Routing Type, 路由类型 (见 5.3.2 节)。
SCNA	16	Source CNA, 源简短网络地址。
DCNA	16	Destination CNA, 目的简短网络地址。
CCI	16	Congestion Control Indicator, 用于传递拥塞控制信息 (见 5.3.5 节)。
LBF	8	Load Balance Factor, 负载均衡因子, 可以作为多路径选择因子使用。LBF 域段的取值由发送端生成, 本规范不做约束。
SL	4	Service Level, 服务优先级, 用于标识链路上数据包的优先级。
Mgmt	1	LPH.CFG = 6: UBFM 特权比特, UBFM 发出的数据包中该 bit 值为 1。 LPH.CFG = 9: 该位无效, 应为 0。
NLP	3	Next Layer Protocol, 指示 NTH 后续的数据包格式  LPH.CFG = 6: <ul style="list-style-type: none"><li>• 3'b000: 为 CTPH, 由 CTPH.NLP 指示后续格式;</li><li>• 3'b001: 为 16-bit UPIH, 用于枚举管理命令。</li><li>• Others: Reserved</li></ul> LPH.CFG = 9: <ul style="list-style-type: none"><li>• 3'b000: 为 TAH, 不带 TPH、EID 和 UPI;</li><li>• Others: Reserved</li></ul>

### 5.2.2 基于 24-bit CNA 地址格式的包头

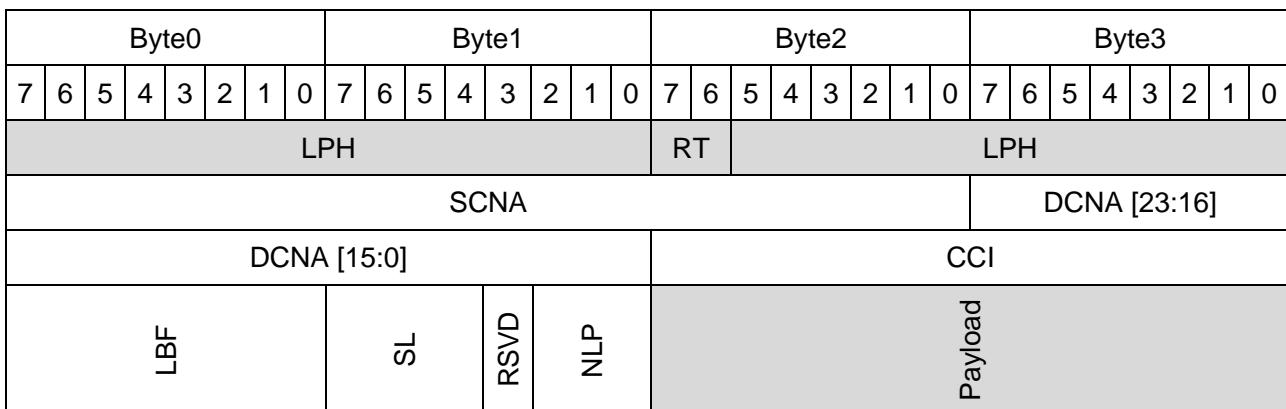


图 5-3 基于 24-bit CNA 地址格式的网络层包头

表 5-2 基于 24-bit CNA 地址格式的网络层包头域段

域段名	位宽(bit)	描述
RT	2	Routing Type, 路由类型 (见 5.3.2 节)。
SCNA	24	Source CNA, 源简短网络地址。
DCNA	24	Destination CNA, 目的简短网络地址。
CCI	16	Congestion Control Indicator, 用于传递拥塞控制信息 (见 5.3.5 节)。
LBF	8	Load Balance Factor, 负载均衡因子。
SL	4	Service Level, 服务优先级, 用于标识链路上数据包的优先级
NLP	3	Next Layer Protocol, 指示 NTH 包头后续的数据包格式 3'b000: 为 CTPH; 3'b010: 为 RTPH; Others: Reserved

### 5.2.3 基于 IP 地址格式的包头

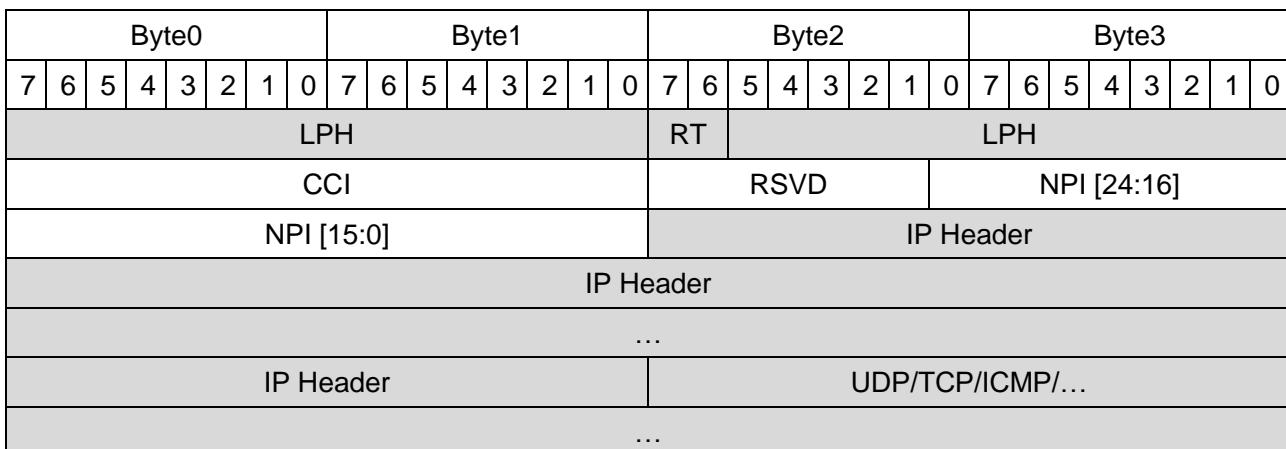


图 5-4 基于 IP 地址格式的网络层包头

UB 网络层可兼容 IP 协议，通过在标准 IP 数据包头部之前添加 UB 网络层域段，可以支持网络隔离、端网协同的负载均衡以及多种拥塞标记模式。基于 IP 地址格式的网络层包头如上图所示，各域段定义如下：

**表 5-3 基于 IP 地址格式的网络层包头域段**

域段名	位宽(bit)	描述
RT	2	Routing Type，路由类型（见 5.3.2 节）。
CCI	16	Congestion Control Indicator，用于传递拥塞控制信息（见 5.3.5 节）。
NPI	25	Network Partition Identifier，网络分区标识（见 5.3.3 节）。

UB 协议栈的通信流量使用 UDP 承载时，目的端口号固定为 4792( IANA Service Name and Transport Protocol Port Number Registry )，对 IP 包头和 UDP 包头部分域段的使用说明如下：

**表 5-4 UB 协议栈使用 IPv4 包头的说明**

IPv4 头域段	支持情况
IHL	设置为 5，不支持 Option 选项
Type of Service	DSCP 根据业务设置优先级； UB Domain 内未使用 ECN 域段，该域段可设置为任意值。
Total Length	IPv4 头到 ICRC 的长度
Flags	设置为 0b010，不支持分片
Fragment Offset	设置为 0，不支持分片
Protocol	设置为 0x11，代表传输层为 UDP

**表 5-5 UB 协议栈使用 IPv6 包头的说明**

IPv6 头域段	支持情况
Traffic class	DSCP 根据业务设置优先级； UB Domain 内未使用 ECN 域段，该域段可设置为任意值。
Payload length	IPv6 头之后到 ICRC 的长度
Next header	设置为 0x11，代表传输层为 UDP

表 5-6 UB 协议栈使用 UDP 包头的说明

UDP header of UB stack	支持情况
Source Port	源端口号，可作为负载均衡因子使用，取值由发送端生成，本规范不做约束
Destination Port	设置为 4792 ( IANA Service Name and Transport Protocol Port Number Registry )
Length	UDP 头到 ICRC 的长度
Checksum	设置为 0

## 5.3 网络层特性

### 5.3.1 网络地址管理

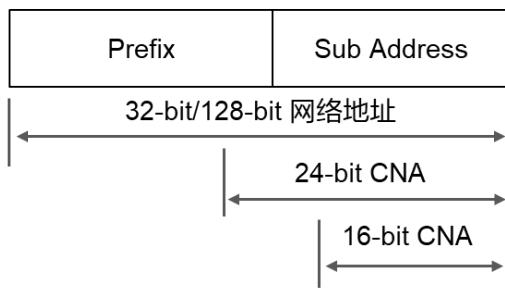


图 5-5 网络地址格式

UB 网络支持 32-bit/128-bit 网络地址，如上图所示，共享一个端口的多个 Entity 可按需使用同一个网络地址，UB 链路上传输的数据包可以携带完整的网络地址格式，也支持携带简短网络地址（CNA）格式，例如：

1. 可使用完整的 32-bit/128-bit 网络地址作为 IP 地址在 Domain 内或跨 Domain 通信，例如 LPH.CFG=3/4 类型数据包（见 B.2 节）。
2. 可使用 16-bit 或 24-bit 简短网络地址在 Domain 内通信，例如 LPH.CFG=6/7/9 类型数据包（见 B.2 节）。

网络地址支持分配给 UB Controller 及其端口使用，也支持分配给 UB Switch 使用，表示其在互连拓扑中的位置。分配给 UB Controller 使用的简短网络地址称为 Primary CNA，分配给 UB Controller 端口使用的简短网络地址称为 Port CNA。

系统管理员应为 UB Domain 规划网络地址前缀，具体划分方式由系统管理员决定。UBFM 根据互连拓扑管理本 Domain 内的网络地址和路由配置。

## 5.3.2 路由机制

### 5.3.2.1 基本路由机制

网络层支持使用全长网络地址和简短网络地址进行路由，UB Switch 和 UB Controller 应支持路由处理单元，本规范不约束路由处理单元的具体实现。如下图所示，UB Switch 或 UB Controller 将目的网络地址输入路由处理单元即可得到一组出端口列表、对应的 Cost (如有) 以及其它厂商自定义信息 (如有)，可选实现的 Cost 代表从每个出端口到达目的地的路径权值 (Cost 值可以通过跳数或带宽表示，或由用户自定义)。UB Controller 或 UB Switch 应根据 Routing Type (RT) 和多路径选择因子从该一组出端口列表中为数据包选择对应的发送端口，其中多路径选择因子由厂商自定义实现。

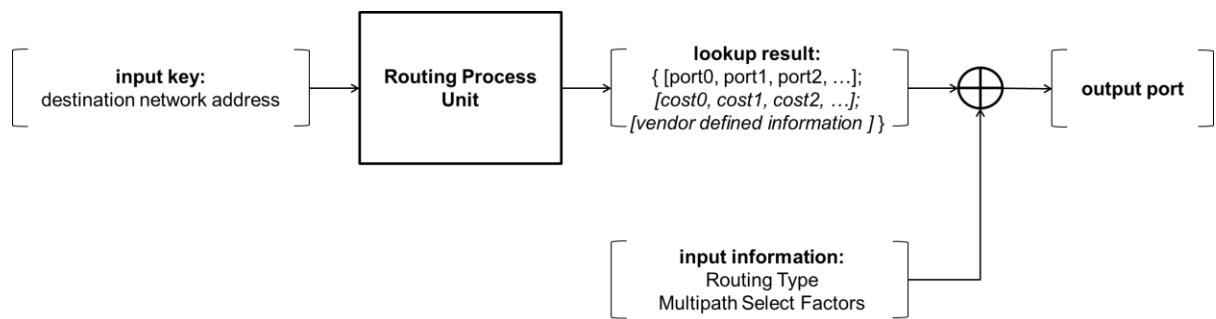


图 5-6 网络层路由机制

RT 是上层协议基于通信任务特征（保序要求、网络利用率等）生成并下发给网络层使用的多路径路由选路信息，UB Controller 或 UB Switch 都可以根据 RT 信息进行负载均衡选路。包头中 RT 域段定义如下：

表 5-7 RT 域段定义

RT 设置 (2 bits)	负载均衡模式	多路径选路策略
00	逐流	基于路由表指示的全部可达路径选路
01	逐包	基于路由表指示的全部可达路径选路
10	逐流	基于路由表指示的最短路径集选路
11	逐包	基于路由表指示的最短路径集选路

当上层协议需要网络层提供保序传输时，应选择逐流负载均衡模式。当上层协议不需要网络层提供保序传输时，可选择逐包负载均衡模式。

### 5.3.2.2 UB Controller 路由

UB Controller (Initiator) 在发送事务请求给 Target 时应为其选择出端口，并基于选择的出端口获取封装网络层包头需要的源网络地址，如下图所示：

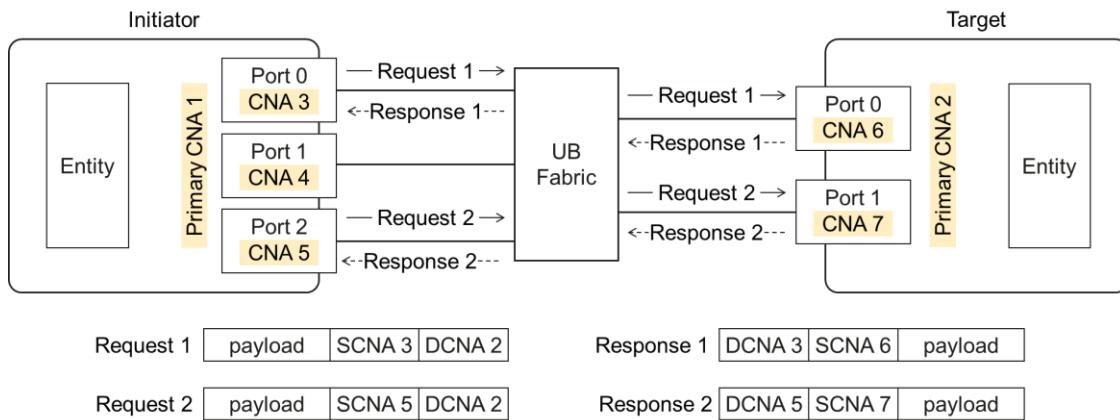


图 5-7 UB Controller 的路由机制示意图

- 当事务请求使用 Target 的 Primary CNA 作为数据包的目的网络地址时，Target 可从其所有端口接收对应的数据包。
- 当事务请求使用 Target 的某个 Port CNA 作为数据包的目的网络地址时，Target 可从该端口接收对应的数据包。
- 事务请求数据包的源网络地址可以使用 Initiator 对应发送端口的 Port CNA 或者实现者为优化设计而设定的其它 CNA。
- 事务响应数据包的目的网络地址可使用对应事务请求数据包的源网络地址，使事务响应数据包能够被发送该事务请求数据包的 Initiator 端口接收。

### 5.3.2.3 UB Switch 路由

一个 UB Switch 可以只配置一个网络地址用于管理和控制，UB Switch 可以从所有端口上接收并捕获目的网络地址是自己的数据包。

## 5.3.3 网络隔离

### 5.3.3.1 NPI

Network Partition 是一组提供 IP 网络接口设备服务的 UB Controller 集合，不同 Network Partition 之间网络通信隔离。Network Partition Identifier ( NPI ) 是 Network Partition 的分区标识。

Permission 1-bit	ID 24-bit
---------------------	--------------

图 5-8 NPI 定义

NPI 域段长度为 25 bits，高 1-bit Permission 是权限位，24-bit ID 代表分区值：

1. 不同 ID 标识的分区间通信隔离。
2. 相同 ID 标识的分区需要进一步通过权限位进行是否可互通的判断，权限位为 0 代表 high 权限，为 1 代表 low 权限。具有 high-high 权限以及 high-low 权限的 UB Controller 之间可以互相通信，具有 low-low 权限的 UB Controller 之间无法互相通信。
3. ID 值为 0 表示无效值，UB Controller 在没有被配置有效 NPI 前，禁止发送 LPH.CFG=3/4 的 IP 数据包。
4. ID 值为 1 表示默认分区。

### 5.3.3.2 UB Controller 的网络隔离

UB Controller 应支持 NPI 的可信配置和防仿冒。发送端发送数据包时添加 NPI，接收端接收数据包时检查 NPI，若 NPI 不匹配，则进行丢弃处理。UB Controller 应支持两种工作模式：严格模式（ Strict Mode ）和宽松模式（ Loose Mode ），如下图所示。

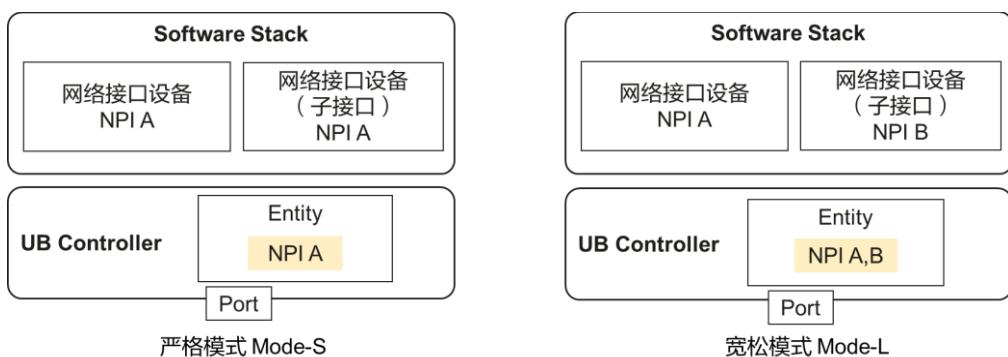


图 5-9 UB Controller 工作模式

1. 严格模式（ Model-S ）： UB Controller 不信任软件栈的网络接口设备。
  - (1) 基于一个 Entity 创建的多个网络接口设备（包括子接口设备）都属于且仅属于一个 NPI。
  - (2) 发送数据包时， NPI A 被封装在包头中， UB Controller 只允许发送携带 NPI A 的数据包。
  - (3) 接收数据包时， UB Controller 进行 NPI 的检查，检查不通过则丢包。
2. 宽松模式（ Model-L ）： UB Controller 信任软件栈的网络接口设备。
  - (1) 基于一个 Entity 创建的多个网络接口设备可以属于不同的 NPI，每个网络接口设备属于且仅属于一个 NPI。
  - (2) 发送数据包时，对应网络接口设备的 NPI 被封装在包头中。
  - (3) 接收数据包时由对应的网络接口设备进行 NPI 检查，检查不通过则丢包。 UB Controller 可辅助进行 NPI 过滤，丢弃 NPI 非法的数据包，拦截拒绝服务攻击。

### 5.3.3.3 UB Switch 的网络隔离

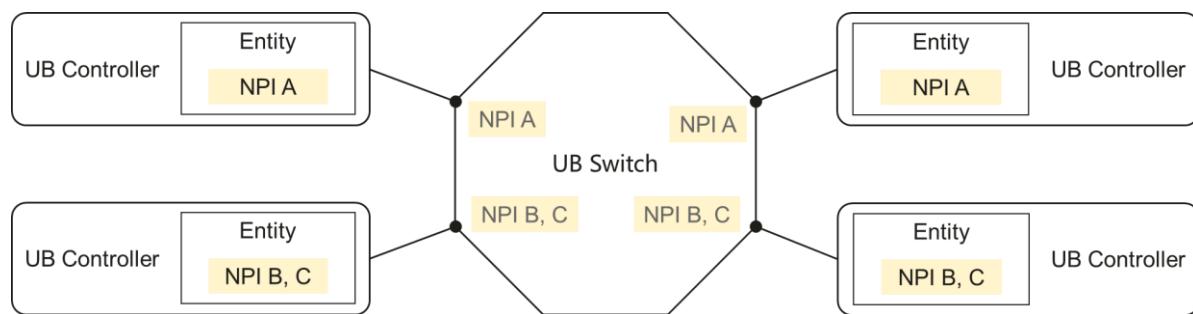


图 5-10 UB Switch 的网络隔离

UB Switch 可支持网络隔离，基于端口配置 NPI 过滤规则，一个端口可加入一个或多个 NPI。边缘 UB Switch 在发送和接收数据包时对 NPI 域段进行检查，丢弃不符合过滤规则的数据包。IP 数据包的目的地为 UB Switch 时，UB Switch 收到数据包后可进行 NPI 检查，防止非法访问或恶意攻击流量。

当管理员不信任端侧时，可启用 UB Switch 的网络隔离功能。UB Switch 可支持分别在发送数据包或接收数据包时启用 NPI 替换功能，将数据包的 NPI 域段替换为配置值。

### 5.3.4 服务质量

网络层使用服务优先级 ( Service Level, SL ) 标识链路上不同数据包的优先级，SL 的取值由具体业务规划，不在本规范定义。

如果实现了多个 VL，UB Controller 应支持 SL-VL 映射和 VL 间调度，UB Switch 应支持 VL 间调度，具体实现方式不在本规范定义。

### 5.3.5 拥塞标记

#### 5.3.5.1 一般要求

UB Controller 和 UB Switch 可支持包括 Confined Active Queue Management ( CAQM ) 和 Forward Explicit Congestion Notification (FECN)在内的多种拥塞标记模式。

- CAQM: CCI.Mode = 3'b000

Byte0								Byte1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Mode		Loc	RSVD	Enable	C	I	Hint								

图 5-11 CAQM 拥塞标记模式

- Enable: CAQM 是否启用,
  - 1: 启用;
  - 0: 未启用。
- Hint: 发送端 UB Controller 请求增加的带宽, 单位可由厂商自定义, UB Controller 和 UB Switch 应保持一致。
- I: 请求确认, UB Switch 的可用带宽若满足请求增加的带宽要求则 I 位置 1, 否则置 0。
- C: 拥塞标记位, 若数据包到达时存在拥塞, 则 C 位置 1。
- Loc: 表示拥塞发生的位置, 用于辅助发送端 UB Controller 的发送控制, 1:接收端边缘 UB Switch; 0: 中间 UB Switch。

## 2. FECN: CCI.Mode = 3'b100

Byte0								Byte1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Mode		Loc	RSVD												FECN

图 5-12 FECN 拥塞标记模式

FECN 域段宽度为 2 bits, 表示拥塞程度:

- (1) 2'b00, 不可标记;
- (2) 2'b01, 可以标记, 表示轻度拥塞;
- (3) 2'b10, 可以标记, 表示无拥塞;
- (4) 2'b11, 可以标记, 表示重度拥塞。

注: 依赖 IP ECN 的协议栈使用 UB 链路收发数据包时应实现 IP 头 ECN 与 CCI 域段 FECN 间的转换, 发送数据包时将 IP 头中的 ECN 拷贝到 CCI 域段中的 FECN, 接收数据包时从 CCI 域段拷贝 FECN 到 IP 头中的 ECN 域段。

## 3. FECN\_RTT: CCI.Mode = 3'b010

Byte0								Byte1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Mode		Loc	Timestamp												FECN

图 5-13 FECN\_RTT 拥塞标记模式

Timestamp: 发送端 UB Controller 填写的时间戳, UB Switch 可不感知, 由接收端 UB Controller 通过 CNP 带回发送端。Timestamp 的精度由发送端决定, 本规范不作约束。

## 4. 保留: CCI.Mode = Others

### 5.3.5.2 FECN 拥塞标记

#### 1. 拥塞检测和标记

UB Switch 应支持根据出端口的拥塞状态对发送的数据包打上 FECN 标记。

- (1) 若 FECN 为不可标记，则不可对数据包进行 FECN 标记，无论是否存在拥塞。
- (2) 若 FECN 为可标记，当本节点的拥塞程度高于数据包携带的标记时，则对数据包进行 FECN 和 Loc 标记，反之则不标记，将网络路径上最拥塞位置的拥塞状态传递到目的端。

#### 2. 拥塞扩散检测和避免

数据链路层信用流控的逐级反压机制可能会导致网络拥塞状态扩散，处于拥塞状态的 UB Switch 并不一定是产生拥塞的源头。UB Switch 可选的支持拥塞扩散检测，只在产生拥塞的源头端口对数据包进行 FECN 打标。该功能应支持由系统管理员配置开启或关闭。

### 5.3.5.3 CAQM 拥塞标记

CAQM 是一种主动队列管理方案，支持拥塞标记和发送带宽增量控制。发送端 UB Controller 在发送数据包时应清除 C 位，UB Switch 应在发生拥塞时将该 C 位置 1。发送端 UB Controller 在发送数据包时可将 I 位置 1 用于请求增加发送带宽，UB Switch 如果判断 Hint 域段指示的增量申请可能导致拥塞产生则将 I 位置为 0，此时 UB Switch 可将本节点的拥塞程度填充到 Hint 域段，该拥塞程度的表达可由厂商自定义。

### 5.3.6 死锁避免

数据链路层信用流控的逐级反压机制可能会造成 UB Switch 缓存的循环依赖，形成死锁，导致整个路径上的所有数据包都无法传输。一个典型的死锁场景如下图所示，节点 D 无法及时处理来自节点 B 的数据包，会反压 UB Switch 4；UB Switch 4 会反压 UB Switch 3；UB Switch 3 会反压节点 C 与 UB Switch 2；UB Switch 2 会反压节点 B 与 UB Switch 1；UB Switch 1 会反压节点 A 与 UB Switch 4；UB Switch 4 又会反压节点 D 与 UB Switch 3，造成死锁。

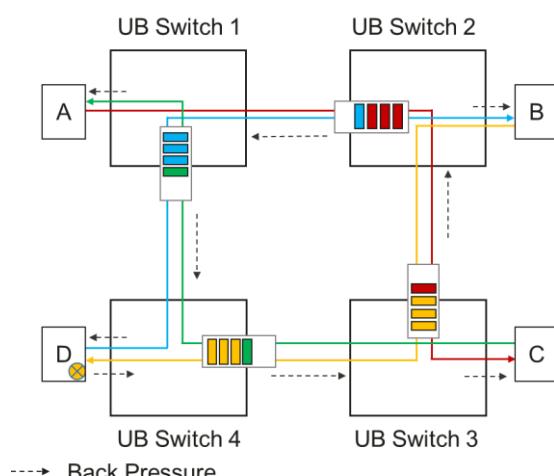


图 5-14 典型死锁场景示意图

可基于以下几种机制的组合实现死锁避免和解除：

1. 路由防死锁：可基于具体的网络拓扑选择特定的路由算法破除死锁形成的条件。
2. VL 主动切换防死锁：在支持多 VL 的网络中，可通过配置网络拓扑特定节点上的 Input VL-Output VL 映射关系，通过强制切换 VL 消除环路依赖，从而避免死锁。
3. 超时丢包解死锁：如果队列超过约定时间都没有得到调度，可认为发生了死锁，此时将队列中的数据包全部丢弃，可打破死锁依赖条件。

### 5.3.7 ICRC

UB Controller 或 UB Switch 在发送数据包时生成 ICRC 并添加到数据包中，在接收到数据包后进行 ICRC 校验，若校验不通过则丢弃数据包。32-bit ICRC 域段位于数据包的尾部，提供端到端保护，覆盖数据包中不可被改变的域段：

1. LPH.CFG=3/4 的数据包，覆盖范围从 IP 头开始；
2. LPH.CFG=6 的数据包，覆盖范围从 NTH 开始；
3. LPH.CFG=7 的数据包，覆盖范围从 NTH 开始；

ICRC 计算原则如下：

1. 采用以太 0X04C11DB7 的 CRC-32 多项式 ( IEEE Std 802.3<sup>TM</sup> - 2022 ) ；
2. 初始值为 0xFFFFFFFF；
3. NTH 中的 CCI 域段和 LBF 域段替换成全 1 进行 ICRC 计算；
4. IPv4 头 ( IETF RFC791 ) 中的 Time To Live、Header Checksum、Type of Service ( DSCP & ECN ) 域段替换成全 1 进行 ICRC 计算；
5. IPv6 头 ( IETF RFC2460 ) 中的 Traffic Class ( DSCP & ECN ) 、Flow Label、Hop Limit 域段替换成全 1 进行 ICRC 计算；
6. UDP 头中 checksum 域段替换成全 1 进行 ICRC 计算；
7. 数据包每个 Byte 内 bit0-bit7 做 bit 翻转后计算 CRC，对 CRC 做 bit 翻转并取反后得到 ICRC，举例如下图所示：

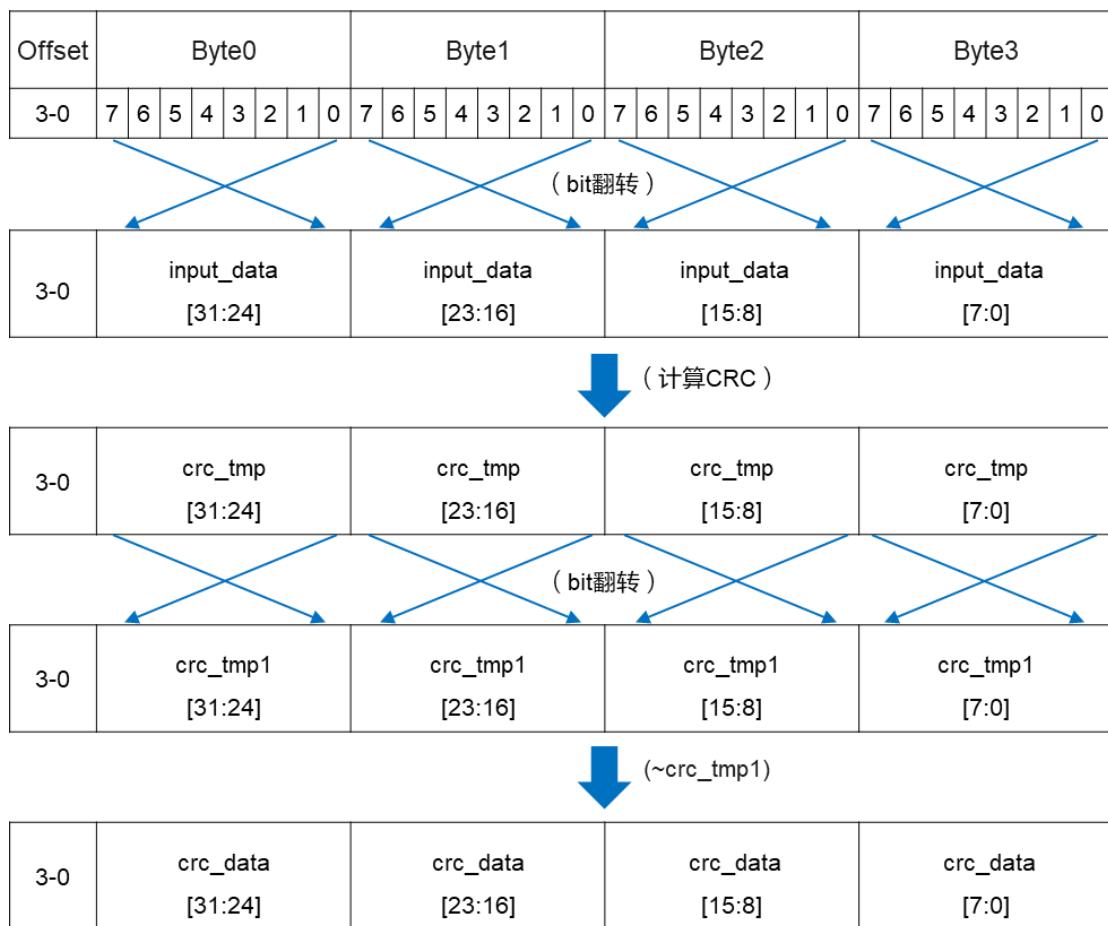


图 5-15 bit 翻转 ICRC 计算示例

### 5.3.8 最大数据包长度

UB Controller 和 UB Switch 支持的最大数据包长度应大于等于数据净荷最大长度（即传输层 MTU）加上链路层及以上所有分层头部和尾部长度的总和，包括 LPH、NTH、TPH、TAH 等数据包头和 ICRC。

# 6 传输层

## 6.1 概述

UB 传输层位于事务层和网络层之间，如图 6-1 所示。UB 传输层基于传输端点( Transport Endpoint, TPEP )为事务层提供端到端可靠/不可靠的传输服务。TPEP 是一种逻辑端点，具备发送和接收传输层 packet 的能力，其中负责发送传输层数据包 ( Transport data Packet, TP Packet ) 的 TPEP 为传输发送方( Transport Sender, TP Sender )，负责接收 TP Packet 的 TPEP 为传输接收方( TP Receiver, Transport Receiver )。TP Sender 接收事务层的事务操作，并负责将其切分并封装为若干个 TP Packet 后下发给网络层。TP Receiver 负责从网络层接收 TP Packet，并在可靠服务下检查事务操作的完整接收。当事务操作接收完成时，传输层负责通知事务层执行。

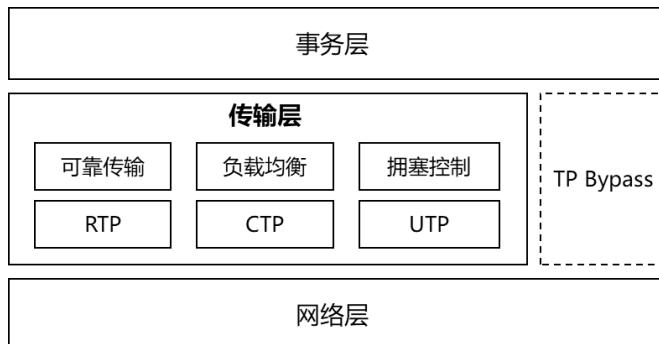


图 6-1 传输层概览

传输层可靠传输服务通过 PSN 机制、重传机制等可靠性机制，保证 TP Packet 能够可靠地送达 TP Receiver，并且保证同一 TP Packet 只会上报事务层执行一次。此外，UB 传输层还提供多路径负载均衡和拥塞控制，降低动态传输时延，实现 TP Packet 的高效传输。不可靠的传输服务不提供重传机制，不保证 TP Packet 送达 TP Receiver。

UB 传输层为事务层提供共享传输通道 ( Transport Channel, TP Channel )，提高可靠传输服务的可扩展性，并可通过建立传输通道组 ( Transport Channel Group, TPG ) 实现一对 UB 处理单元 ( UB Processing Unit, UBPU ) 间多个 TP Channel 的统一管理。UB 事务层在使用可靠传输服务之前，需要为 TP Sender 与 TP Receiver 建立 TP Channel。TP Channel 可以被多个 Initiator 和 Target 对共享，用以减少 TP Channel 建立和维护的资源开销。同时，一对 UBPU 之间可以建立多个 TP Channel，多个 TP Channel 可组成 TPG。TPG 向事务层呈现为一个整体，统一为事务层提供传输服务，负责若干个事务操作在多个 TP Channel 之间的负载均衡。

此外，UB 传输层的设计支持与细粒度负载均衡算法（如逐包负载均衡等）的适配，以提高整网利用率。具体地，UB 传输层支持使能乱序接收。

为适应事务层差异化的需求，传输层（Transport, TP）提供了可选择的传输模式，分别是可靠传输模式（Reliable Transport, RTP）、轻量级传输模式（Compact Transport, CTP）和不可靠传输模式（Unreliable Transport, UTP）。此外，传输层还可配置为旁路模式（Transport Bypass, TP Bypass）。四种模式支持的传输层特性如表 6-1 所示。RTP 为事务层提供可靠传输服务，CTP 联合下层协议栈为事务层提供可靠传输服务，UTP 为事务层提供不可靠的传输服务，TP Bypass 不提供传输层服务。

1. RTP 提供端到端传输服务，以及基于 TP Channel 的负载均衡和拥塞控制机制，一般用于对可靠性要求较高的场景。RTP 可通过 TP Channel 被多个 Initiator 和 Target 对共享，提高可扩展性。
2. UB 协议要求 CTP 联合其下层协议栈，为事务层提供可靠传输服务。CTP 提供粗粒度拥塞管理机制：提供基于 Entity 的多路径负载均衡，以及基于目的 Entity 和 VL（Virtual Lane）粒度的拥塞控制机制。CTP 一般用于链路质量较高或者 UBPU 直连的场景。
3. UTP 提供不可靠的传输服务，一般用于对丢包不敏感的业务如带内建立连接等业务。
4. TP Bypass 无传输层特性，无传输层协议栈开销，一般配合 Load/Store 同步访问内存事务使用。

**表 6-1 不同传输模式下支持的特性**

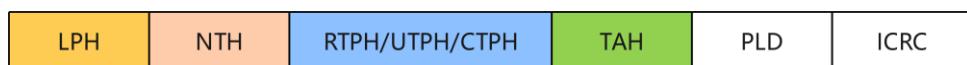
传输模式	对事务层可靠性	多路径负载均衡	拥塞控制	其它特性
RTP	提供可靠传输服务	基于 TP Channel	基于 TP Channel	TPG 管理、 TP Channel 共享
CTP	联合下层协议栈提供可靠传输服务	基于 Entity	基于目的 Entity 和 VL	/
UTP	提供不可靠的传输服务	/	/	/
TP Bypass	不涉及	不涉及	不涉及	不涉及

## 6.2 传输层 packet 格式

传输层 packet 包括传输层数据包 TP Packet 和传输层应答包。TP Packet 由 TP Sender 生成，携带了事务请求或响应；传输层应答包由 TP Receiver 生成，用于对 TP Packet 的应答（仅限 RTP）及拥塞通知（仅限 RTP/CTP）。传输层应答包的 CFG 及 NTH 类型，和其对应的 TP Packet 一致。

RTP 使用 RTPH（Reliable Transport Header）包头，UTP 使用 UTPH（Unreliable Transport Header）包头，CTP 使用 CTPH（Compact Transport Header）包头，TP Bypass 没有传输层包头。网络层包头的 NLP 域段指示传输层使用的包头类型或指示使用 TP Bypass 模式。RTPH/UTPH/CTPH 是传输层的基本头。

传输层数据包 TP Packet 格式：



**图 6-2 TP Packet 格式**

传输层应答包可携带不同的扩展包头，格式如下，其中 TPACK/TPACK-CC/TPNAK/TPNAK-CC/TPSACK/TPSACK-CC 仅在 RTP 下使用：

(a) 基本的传输层应答包 ( TPACK/TPNAK ) :



图 6-3 TPACK/TPNAK 格式

(b) 携带拥塞反馈信号的传输层应答包 ( TPACK-CC/TPNAK-CC ) :



图 6-4 TPACK-CC/TPNAK-CC 格式

(c) 选择性重传传输层应答包 ( TPSACK ) :



图 6-5 TPSACK 格式

(d) 带拥塞控制的选择性重传传输层应答包 ( TPSACK-CC ) :

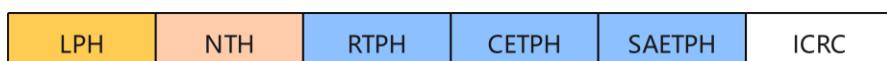


图 6-6 TPSACK-CC 格式

(e) 单独的拥塞通知包 ( CNP ), 不具备对 TP Packet 的应答能力, 仅携带拥塞信息, 在 RTP/CTP 下使用:

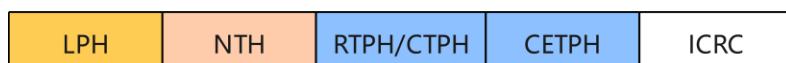


图 6-7 CNP 格式

## 6.2.1 Transport Header ( RTPH/UTPH/CTPH )

RTPH 格式如下：

	Byte0								Byte1								Byte2								Byte3													
	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						
0	TPOopcode								TPVer	Padding		NLP				SrcTPN[23:8]																						
4	SrcTPN[7:0]								DstTPN																													
8	A	F	Reserved						PSN																													
12	RSPST		RSPINFO						TPMSN																													

图 6-8 RTPH 格式

UTPH 格式如下：

	Byte0								Byte1								Byte2								Byte3													
	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						
0	TPOopcode								TPVer	Padding		NLP				Reserved																						
4	Reserved								Reserved																													
8	Reserved								Reserved																													
12	Reserved								Reserved																													

图 6-9 UTPH 格式

CTPH 格式如下：

Byte0							
7	6	5	4	3	2	1	0
TPOopcode				Padding			

图 6-10 CTPH 格式

表 6-2 RTPH/UTPH/CTPH 域段

域段名	位宽(bit)	RTPH	UTPH	CTPH
TPOopcode	8(RTPH), 8(UTPH), 2(CTPH)	<p>Transport Opcode[7:0], 用于指示 packet 类型以及当前 TP Packet 是否为事务操作的最后一个 TP Packet。</p> <p>TPOopcode[6:0]:</p> <ul style="list-style-type: none"> <li>• 0x0: UTP 下使用;</li> <li>• 0x1: Reliable TP Packet, RTP 下使用, 下一层包头由 TPH 中的 NLP 域段指定;</li> <li>• 0x2: TPACK/TPNAK, 传输层应答包, 无下一层包头;</li> <li>• 0x3: TPACK-CC/TPNAK-CC, 带有拥塞信息的传输层应答包, 下一层包头为 CETPH;</li> <li>• 0x4: Reserved;</li> <li>• 0x5: TPSACK, 带有选择性确认信息的传输层应答包, 下一层包头为 SAETPH;</li> <li>• 0x6: TPSACK-CC, 带有拥塞</li> </ul>	<p>TPOopcode[0:0]:</p> <ul style="list-style-type: none"> <li>• 0x0: Unreliable TP Packet, UTP 下使用, 下一层包头由 TPH 中的 NLP 域段指定;</li> <li>• 其它域段在 RTP 下使用。</li> </ul>	<p>Transport Opcode[1:0], 用于指示是否为 CNP。</p> <p>TPOopcode[1:0]:</p> <ul style="list-style-type: none"> <li>• 0x0: CTP 数据包;</li> <li>• 0x1: CNP;</li> <li>• Others: Reserved。</li> </ul> <p>CTP 数据包下一层包格式由 CTPH 中的 NLP 域段指定。</p>

域段名	位宽(bit)	RTPH	UTPH	CTPH
		<p>信息和选择性确认信息的传输层应答包，下一层包头为 CETPH，CETPH 后还跟有 SAETPH；</p> <ul style="list-style-type: none"> <li>• 0x7: Reserved;</li> <li>• 0x8: CNP，拥塞通知包，下一层包头为 CETPH。</li> <li>• Others: Reserved。</li> </ul> <p>TPOopcode[7]用于表示当前 TP Packet 是否为事务操作的最后一个 TP Packet ( 记为 last 标记 )，与 TPMSN ( 见下文 TPMSN 域段解释 ) 一起使用，可用于判断事务操作的结束，仅在 TPOopcode[6:0]为 0x0 或 0x1 时有效：</p> <ul style="list-style-type: none"> <li>• 0x0: 非当前事务操作的最后一个 TP Packet;</li> <li>• 0x1: 当前事务操作的最后一个 TP Packet。</li> </ul>		
NLP	4	<p>TPOopcode[6:0]==0x0 或 0x1 时，NLP ( Next Layer Protocol ) 用于指示下一层包头类型：</p> <ul style="list-style-type: none"> <li>• 0x0: ATAH/BTAH ( 见 7.2 节 )；</li> <li>• 0x1: 32bits UPIH + 256bits EIDH ( 见附录 B )；</li> <li>• 0x2: Reserved;</li> <li>• 0x3: CIPH ( 见 11.5 节 )；</li> <li>• Others: Reserved。</li> </ul>	见 RTPH	<p>用于指示下一层包头类型：</p> <ul style="list-style-type: none"> <li>• 0x0: ATAH/BTAH;</li> <li>• 0x1: 32bits UPIH + 256bits EIDH+TAH;</li> <li>• 0x2: 16bits UPIH + 40bits EIDH+TAH;</li> <li>• 0x3: CIPH;</li> <li>• Others: Reserved。</li> </ul>
Padding	2	从 TPH 第一个字节开始到 Payload 最后一个字节 ( Payload 可为 0 ) 的长度需要保持 4 字节对齐，该域段表示附加到传输层 packet 末尾的 0 填充字节数。	见 RTPH	见 RTPH

域段名	位宽(bit)	RTPH	UTPH	CTPH
TPVer	2	Transport 协议版本号, 本规范支持的值为 0。	见 RTPH	无此域段
SrcTPN	24	源 TPEP 编号。	无此域段	无此域段
DstTPN	24	目的 TPEP 编号。	无此域段	无此域段
A	1	Ack required, 表示当前 TP Packet 是否要求传输层 TP Receiver 返回单独应答。为 1 时表示当前请求需要返回单独应答；为 0 时不需要返回单独应答。	无此域段	无此域段
F	1	1 表示当前 TP Packet 为 Fake 包, 0 表示当前 TP Packet 为普通包。TP Sender 在收到携带事务层资源不足的传输层应答时发送 Fake 包用于 TP 连接保活（见 8.2.7.3 节）。	无此域段	无此域段
PSN	24	Packet Sequence Number, 应符合 6.4.1 节相关要求。	无此域段	无此域段
TPMSN	24	TP Message Sequence Number。	无此域段	无此域段
RSPST	3	<p>Response Status。</p> <ul style="list-style-type: none"> <li>• 对于 TPACK/TPNAK, 表示事务层完成状态:           <ul style="list-style-type: none"> <li>- 3'b000: Successful, 表示 TPACK。</li> <li>- 3'b001: 携带事务层 RNR(Receiver Not Ready), 此时 RSPINFO 表示 RNR_Timer; 当接收端资源不足时, 返回 RNR。</li> <li>- 3'b010: 携带事务层 Page Fault, 此时 RSPINFO 表示 RNR_Timer, 当出现 Page Fault 时使用。</li> <li>- 3'b011: 携带事务层 Completer Error。</li> <li>- 3'b100: Reserved。</li> </ul> </li> </ul>	无此域段	无此域段

域段名	位宽(bit)	RTPH	UTPH	CTPH
		<ul style="list-style-type: none"> <li>- 3'b101: 表示 TPACK, 携带事务层应答信息, 具体信息由 RSPINFO 指定。TP Sender 接收到此应答时, TPACK 透传事务层作为对事务操作的应答。</li> <li>- Others: Reserved。</li> </ul> <ul style="list-style-type: none"> <li>• 对于 TPSACK, RSPST 和 RSPINFO 域段固定为 0。</li> </ul>		
RSPINFO	5	<p>此域段对应答包有效, 对其它 packet 为 Reserved。特别地, TPSACK 此域段固定为 0。</p> <ul style="list-style-type: none"> <li>• RSPST=3'b001 和 3'b010 时: 该域段表示 RNR_Timer 编码。RNR_Timeout = <math>2 \mu s * 2^{RNR\_Timer}</math>, RNR_Timer 有效编码范围为[0, 19], 其它编码 Reserved。</li> <li>• RSPST=3'b011 时: 携带错误应答, 错误类型为: <ul style="list-style-type: none"> <li>- 5'b00000 表示 PSN Error; TP Receiver 不支持乱序接收时才会有该错误类型。</li> <li>- 5'b00001 携带事务层 Unsupported Request。</li> <li>- 5'b00010 携带事务层 Remote Abort。</li> <li>- Others: Reserved。</li> </ul> </li> <li>• RSPST=3'b100: Reserved。</li> <li>• RSPST=3'b101: 携带事务层应答, 携带的应答类型如下: <ul style="list-style-type: none"> <li>- 5'b00000 表示异常 TAACK, 类型为 TA 重传可恢复。</li> <li>- 5'b00001 表示异常 TAACK, 类型不可恢复错误。</li> <li>- 5'b00010 表示正确应答 TAACK。</li> </ul> </li> </ul>	<p>无此域段</p> <p>无此域段</p>	

字段名	位宽(bit)	RTPH	UTPH	CTPH
		<ul style="list-style-type: none"> <li>- Others: Reserved。</li> <li>• Others: Reserved。</li> </ul>		

### 6.2.2 Congestion Extended Transport Header ( CETPH )

TP Receiver 发送至 TP Sender 的拥塞控制扩展传输头 ( CETPH ), 用于实现拥塞控制。由 TPACK-CC、TPNAK-CC、TPSACK-CC 携带, 或由 CNP 单独携带。

UB 可选支持多种拥塞控制算法 ( LDCP, CAQM, DCQCN 等, 见 6.6 节 ), 拥塞控制算法类型在建立 TP Channel 时进行协商 ( 具体实现方式不在本规范定义 )。

	Byte0								Byte1								Byte2								Byte3							
	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
0	Ack_Seq																															
4	Ce_Seq																															

图 6-11 LDCP CETPH 格式图

表 6-3 LDCP CETPH 域段

字段名	位宽(bit)	描述
Ack_Seq	32	Ack Sequence, 表示 TP Receiver 收到的 Sequence 累计数, 以 2 字节为单位, 即 1 Sequence=2 Bytes; 特别地, 数据包大小不被 Sequence 整除时, 以 Sequence 为单位向上对齐。TP Receiver 每收到一个 TP Packet, 累加此 TP Packet 的 Sequence 数。
Ce_Seq	32	Congested Sequence, 表示 TP Receiver 收到含拥塞标记的累计 Sequence 数, 以 2 字节为单位。数据接收端每收到一个携带拥塞标志的 TP Packet, 累加此 TP Packet 的 Sequence 数。

	Byte0								Byte1								Byte2								Byte3							
	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
0	Ack_Seq																															
4	Reserved								Loc	I	C								Hint													

图 6-12 CAQM CETPH 格式

表 6-4 CAQM CETPH

字段名	位宽(bit)	描述
Ack_Seq	32	同 LDCP CETPH 描述。

字段名	位宽(bit)	描述
Loc	1	表示拥塞发生的位置，该字段由收到的 TP Packet 中网络层包头的 Loc 域段复制而来。 <ul style="list-style-type: none"><li>• 0：网络中间节点拥塞。</li><li>• 1：最后一跳 UB Switch 出口拥塞。</li></ul>
I	1	I ( Increase )：该字段由收到的 TP Packet 中网络层包头的 Increase 域段复制而来。如果网络允许增加发送量，该位为 1，否则为 0。
C	8	C ( Congestion )：表示从上次回复 TPACK 后，TP Receiver 收到的 C 位被置位的 TP Packet 数量，反馈给 TP Sender。
Hint	16	该字段由收到的 TP Packet 中网络层包头的 Hint 域段复制而来。当多个 TPACK 被聚合时（一个 TPACK 确认多个 TP Packet），该字段表示自从上次返回 TPACK 后，所有被允许的拥塞窗口增量之和。

CNP ( Congestion Notification Packet ) 是单独的拥塞通知包，其 CETPH 包头中的 ECN 字段由 TP Packet 网络层包头中携带的 FECN 字段值决定（见网络层 5.3.5.2 节）。CETPH 中的 ECN 字段不会被 UB Switch 修改。

	Byte0								Byte1								Byte2								Byte3							
	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
0	ECN	Loc															Reserved															
4																	Reserved															
8																	Reserved															
12																	Reserved															

图 6-13 CNP CETPH 格式

表 6-5 CNP CETPH

域段名	位宽(bit)	RTP	CTP
ECN	2	拥塞程度指示。在启用 DCQCN 拥塞控制算法时使用。UB 可选支持对拥塞程度的区分，但对拥塞状态的标记必选支持。 <ul style="list-style-type: none"><li>• 0x0：无拥塞；</li><li>• 0x1：表示轻度拥塞；</li><li>• 0x2：reserved；</li><li>• 0x3：表示重度拥塞。</li></ul>	同 RTP

字段名	位宽(bit)	RTP	CTP
Loc	1	表示拥塞发生的位置。UB 可选支持。 <ul style="list-style-type: none"><li>• 0: 网络中间节点拥塞;</li><li>• 1: 最后一跳 switch 出口拥塞。</li></ul>	同 RTP

### 6.2.3 Selective Acknowledge Extended Transport Header ( SAETPH )

用于携带 TP Receiver 接收 TP Packet 的 BitMap 等信息，用在 TPSACK, TPSACK-CC 传输层应答包中。

	Byte0								Byte1								Byte2								Byte3							
	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
0	Reserved			BitMap Size		MaxRcvPSN																										
4	BitMap0																															
8~32	.....																															

图 6-14 SAETPH 格式

表 6-6 SAETPH

字段名	位宽(bit)	描述
BitMapSize	3	TPSACK 或 TPSACK-CC 包携带的 BitMap 的 bit 个数。 <ul style="list-style-type: none"><li>• 0x0: 64bits;</li><li>• 0x1: 128bits;</li><li>• 0x2: 256bits;</li><li>• 0x3: 512bits;</li><li>• 0x4: 1024bits;</li><li>Others: Reserved。</li></ul>
MaxRcvPSN	24	TP Receiver 接收到的最大 PSN。
BitMap	N ( 等于 BitMapSize 表示的 bit 数量 )	一个 bit 对应一个 PSN。RTPH.PSN 是 BitMap[0] 表示的 PSN。 <ul style="list-style-type: none"><li>• bit 为 1, 表示收到了该 PSN 对应的 TP Packet。</li><li>• bit 为 0, 表示未收到该 PSN 对应的 TP Packet。</li></ul>

## 6.3 传输层模式

### 6.3.1 RTP 传输层模式

RTP ( Reliable Transport ) 提供可靠传输服务。可靠传输服务基于传输端点 ( Transport Endpoint, TPEP ) 为事务层提供端到端可靠传输服务，保证传输发送方 ( Transport Sender, TP Sender ) 发送的 TP Packet 可靠地送达传输接收方 ( Transport Receiver, TP Receiver )，并确保同一 TP Packet 只会上报事务层执行一次 ( exactly once )。

RTP 提供的可靠传输服务包括 PSN 机制和 TP Packet 重传机制在内的可靠传输机制 ( 见 6.4 节 )。TP Packet 通过 packet 序列号 ( Packet Sequence Number, PSN ) 进行编号，PSN 唯一标识了 TP Packet 在属于同一 TP Channel 的一组 TP Packet 中的位置，是可靠传输机制实现的前提。TP Sender 和 TP Receiver 进行 TP Packet 传输前，需协商一致建立 TP Channel 并且在传输过程中维护数据传输的上下文，TP Channel 建立流程的具体实现方式不在本规范定义。TP Receiver 根据所接收到的 PSN，检测 TP Packet 的 PSN，并对 TP Sender 回复相应的应答包。应答包类型包括传输正确应答 ( Transport Acknowledgement, TPACK )、传输错误应答 ( Transport Negative Acknowledgement, TPNAK ) 和传输选择性应答 ( Transport Selective Acknowledgement, TPSACK )。TP Sender 根据收到的应答包更新 TP Packet 的接收状态；若 TP Sender 检测到丢包，根据所启用的重传算法进行重传。

RTP 提供的可靠传输服务流程如图 6-15 所示。

1. TP Sender 发送携带 PSN 的 TP Packet，PSN 根据发送的非重传 TP Packet 数量依次递增。
2. TP Receiver 根据收到的 TP Packet 的 PSN 判断该 TP Packet 是重复包、正序包、乱序包还是位于无效区间的包，并通过向 TP Sender 发送应答包 TPACK/TPNAK/TPSACK 告知 TP Packet 送达情况。
3. TP Sender 根据应答包 TPACK/TPNAK/TPSACK 信息更新已发送 TP Packet 的送达情况，如果收到 TPNAK 或 TPSACK，还需要重传发生丢失的 TP Packet。

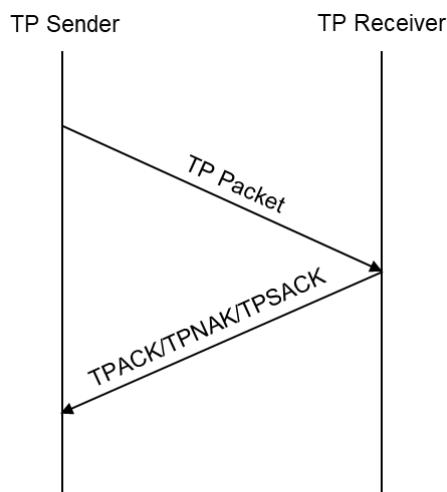


图 6-15 可靠传输服务流程

特别地，RTP 传输模式提供捎带事务层应答的能力，供事务层 ROI、ROT 和 ROL 模式（见 7.3.1 节相关要求）使用。具体地，事务层 ROL 模式下，传输层应答包承载事务层 TAACK 应答。此外，传输层应答可以携带事务层响应 TAACK，或 RNR、Page Fault 等事务层错误信息（事务层应答类型由 RTPH 中的 RSPST 和 RSPINFO 域段联合指定）。

RTP 提供基于 TP Channel 粒度的 UBPU 多端口间负载均衡（见 6.5.1 节）和拥塞控制机制（见 6.6.1 节）。RTP 的 TP Channel 可被属于多个 Initiator 和 Target 对的不同事务操作共享。

### 6.3.2 CTP 传输层模式

UB 协议要求 CTP 联合其下层协议栈，对事务层提供可靠传输服务。本规范中描述一种 CTP 实施例，CTP 可将可靠传输机制卸载至下层协议栈：CTP 不生成传输层应答包和提供重传机制，配合链路层提供的点到点重传功能，在 UBPU 直连场景或链路质量较高场景下，为事务层提供可靠传输服务。在此场景下，packet 丢失可能性低，因此不需要使用开销较大的端到端重传机制保证可靠性。

CTP 还提供粗粒度的拥塞管理机制（见 6.5.2 节和 6.6.2 节）。

### 6.3.3 UTP 传输层模式

UTP 的不可靠传输服务提供了无连接、尽力而为的传输服务，不保证 TP Sender 发送的 TP Packet 可以正确到达 TP Receiver。UTP 无需建立 TP Channel，不生成传输层应答包，也不支持重传机制。

## 6.4 RTP 可靠传输机制

### 6.4.1 PSN 机制

UB 可靠传输服务使用 PSN（Packet Sequence Number）机制检测 TP Packet 是否为正序包。PSN 是为 TP Sender 发送的每个 TP Packet 分配的单调递增序列号，其唯一标识了 TP Packet 在隶属同一 TP Channel 的 TP Packet 中的位置。特别地，事务层 ROL 模式下 PSN 还承担事务保序的功能。

TP Sender 负责对非重传 TP Packet 进行 PSN 的顺序编号，TP Sender 发出的 TP Packet 携带的 PSN 标识了其序列号；TP Receiver 发出的 TPACK 和 TPSACK 应答包携带的 PSN 标识了被正序接收 TP Packet 的最大序列号，TPNAK 应答包携带的 PSN 标识了 TP Receiver 期望接收的 PSN（Expected PSN，EPSN），其中 TP Packet 特指 UB 传输层根据事务层下发的操作拆分成的数据包。建立 TP Channel 时，TP Sender 和 TP Receiver 双方协商 PSN 初始值，PSN 初始值在 0~(16M-1)之间随机。每个 TP Channel 独立维护 PSN。

TP Sender 维护 PSN，初始值设置为协商后的 PSN 值，TP Sender 在发送的 TP Packet 的 RTPH.PSN 域段中携带 TP Sender 所维护的 PSN，每顺序发出一个 TP Packet 后，其本地维护的 PSN 值增加 1。TP Packet 以外的其它 packet（即传输层应答包，如 TPACK 等）不占用 PSN，不会导致 PSN 的增加。TP

Receiver 维护期望收到的 EPSN，EPSN 为 TP Receiver 期望收到的下一个 TP Packet 的 PSN，即期望收到的正序 PSN，其初始值为双方协商的 PSN 初始值。TP Receiver 接收 TP Packet 时，根据 TP Packet 的 PSN 和所维护的 EPSN 判断此 TP Packet 为正序包还是非正序包。TP Receiver 检查 TP Packet 携带的 PSN 与其维护的 EPSN 是否相等；若相等，说明此 TP Packet 为正序包，TP Receiver 接收该 TP Packet，EPSN 增加 1；若不相等，则说明此 TP Packet 非正序包，EPSN 不变。TP Receiver 收到非正序包时，需要判断 TP Packet 落在重复区间、乱序区间、还是无效区间。重复区间表示 TP Receiver 先前已经接收到的 TP Packet 所在的区间，TP Receiver 丢弃该 TP Packet。当 TP Receiver 使能乱序接收时，需设置乱序区间，乱序区间范围起始值为 EPSN+1，区间大小为所设置的最大允许的乱序程度；落在乱序区间内的 TP Packet 被正常接收。EPSN、重复包区间和乱序区间之外的 PSN 区间为无效区间，TP Receiver 丢弃该 TP Packet。

PSN 位宽 24bits，其表示范围为 0~16M-1。当 PSN 超过 24bits 表示范围时，高位溢出。TP Sender 被允许发出的 TP Packet 的最大 PSN 减去等待应答的最小 PSN 应不超过 8M-1。以下举例说明：

1. 等待应答的最小 PSN=0，允许发送的最大 PSN=8M-1 ( 8,388,607 );
2. 等待应答的最小 PSN=8M ( 8,388,608 )，允许发送的最大 PSN=16M-1 ( 16,777,215 );
3. 等待应答的最小 PSN=10M ( 10,485,760 )，允许发送的最大 PSN=18M-1 ( 18,874,367 )，18M-1 超过 24bits 的表示范围，高位溢出，即允许发送的最大 PSN=2M-1 ( 2,097,151 )。

PSN 区间图如图 6-16 所示。[EPSN-8M,EPSN-1] 为最大重复包区间。乱序区间大小可被配置为 [128,256,512,1024,2048] 中的值。



图 6-16 PSN 区间图

#### 示例：PSN 使用示例

事务操作和事务响应（如 TAACK）在 TP Sender 拆分并封装成 TP Packet 进行传输，并共用一套 PSN，传输层应答包 TPACK/TPNAK/TPSACK 不属于 TP Packet 范畴，不与 TP Packet 共用 PSN。如图 6-17 所示，UBPU A 和 UBPU B 通过一个 TP Channel 互发流量，A 和 B 分别维护一套 PSN，UBPU A 往 UBPU B 发送 TP Packet 时，使用 APSN 设置 PSN，初始化为 10；类似地，UBPU B 往 UBPU A 发送 TP Packet 时使用 BPSN，初始化为 0。为了突出 PSN 的使用，图中省略了事务层和传输层的交互过程，并且假设 Write 操作长度正好为一个 TP Packet。UBPU A 往 UBPU B 发送承载事务层 Write 操作的 TP Packet，携带 PSN 为 APSN 的初始值 10，当 UBPU B 成功收到此 TP Packet 后，回复 TPACK 应答且应答中设置的 PSN 为 10。由于 Write 操作仅被拆分成单个 TP Packet，该 TP Packet 的成功接收将直接触发事务操作的完

成。UBPU B 的事务层处理完该操作后, 会回复 TAACK, 此 TAACK 在传输层会被封装成一个 TP Packet, 其携带的 PSN 为 BPSN 的初始值 0。UBPU B 往 UBPU A 发送承载事务层 Write 操作的 TP Packet 过程与上述过程类似。

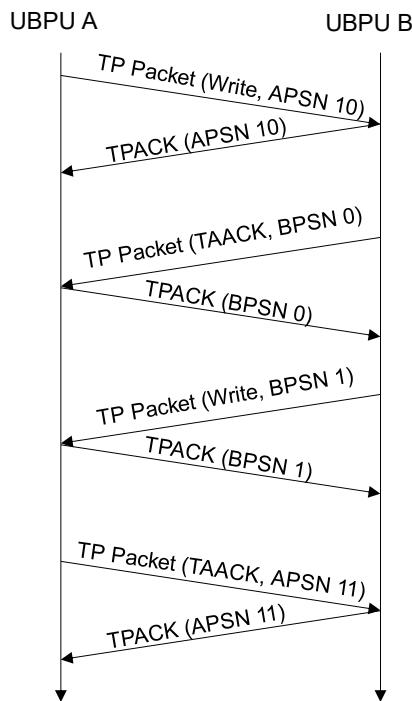


图 6-17 PSN 使用示例

## 6.4.2 重传机制

UB 传输层支持 GoBackN 重传和选择性重传两种重传机制, 其规定了重传的 TP Packet 范围, 其中选择性重传 UB 传输层可选支持。在两种重传机制下, 依据重传触发条件又可以配合快速重传或超时重传。UB 使用快速重传作为重传触发条件时, 在 TP Sender 收到错误应答后, 立即进行重传; 超时重传则需 TP Sender 等待重传超时时间 (Retransmission Timeout, RTO) 后才启动重传。超时重传必须启用, 快速重传可选启用; 在两者同时启用时, 超时重传作为快速重传的兜底机制存在: 例如在尾包丢失发生时, 会触发超时重传。因为超时重传默认启用, 为了避免混淆, 本章节中将 UB 支持的重传算法分为 GoBackN 重传-配合快速重传、GoBackN 重传-无快速重传、选择性重传-配合快速重传和选择性重传-无快速重传, 分别进行描述。

UB 传输层使用的重传算法可由用户配置, 且上述四种重传算法适用的场景如表 6-7 所示。当路由可能会引起乱序接收时 (即通过同一 TP Channel 传输的 TP Packet 在网络中使用多路径传输, 如使用逐包负载均衡), 为了避免不必要的重传, 一般不使用快速重传作为重传触发条件。

表 6-7 重传算法推荐使用场景描述

重传算法	适用场景	
	负载均衡算法	网络丢包率
GoBackN 重传-配合快速重传	同一 TP Channel 的 TP Packet 在网络中使用单路径传输，如逐流负载均衡。	非常低
GoBackN 重传-无快速重传	同一 TP Channel 的 TP Packet 在网络中可能使用多路径传输，如逐包负载均衡。	非常低
选择性重传-配合快速重传	同一 TP Channel 的 TP Packet 在网络中使用单路径传输，如逐流负载均衡。	低
选择性重传-无快速重传	同一 TP Channel 的 TP Packet 在网络中可能使用多路径传输，如逐包负载均衡。	低

#### 6.4.2.1 重传触发条件

UB 使用快速重传作为重传触发条件时，在 TP Sender 收到错误应答后，立即进行重传；超时重传则需 TP Sender 等待 RTO 后才启动重传，重传未被确认接收的 TP Packet。本章节详细描述超时重传的详细设计。

示例：超时重传计时器设置逻辑示例

在一轮计时流程中，发送第一个新包时启动计时器，收到 TPACK/TPSACK 应答包携带的 PSN 大于等于 TP Sender 最小未被确认的 TP Packet PSN 时重置计时器，在确认此轮所有 TP Packet 接收成功后关闭计时器，否则在达到所设定的超时时间后进行重传并重置计时器。

UB 传输层可选两种超时方式，分别为静态超时和动态超时，其中静态超时时间配置后固定不变，动态超时的超时时间在连续超时重传时，随重传次数动态变化。

示例：静态超时时间配置示例

静态超时时间在建立 TP Channel 时可配置，配置的时间阈值可在[512 μ s, 16ms, 128ms, 4s]中选择。

动态超时时间采用指数退避的原则，下一次重传间隔是上一次重传间隔的  $2^N$  倍。动态超时重传的超时间隔计算公式：  
 $RTO = Base\_time * 2^{(N*Times)}$ 。Base\_time 表示首次超时时间间隔，Base\_time 取值范围为 4~2097152，单位 μ s；一般建议 Base\_time 设置为和 RTT ( Round-Trip Time ) 正相关，UB 支持通过测量得到 RTT；N 为用户可配置的超时间隔系数；Times 表示当前已超时重传次数，Times 初始值为 0，每超时重传一次 Times 加一，最大允许重传次数可设置，连续超时重传次数超过设置的最大重传次数时，判定为路径不可达，UB 支持上报携带重传超过次数的错误类型 CQE 到事务层。

示例：动态超时时间配置示例

设置 N=3，最大重传次数=7，第 1 次超时重传周期设置为 20 μ s，则 7 次重传的间隔如表 6-8 所示。

表 6-8 动态超时重传间隔表

重传次数	超时间隔 ( μ s )
1	20
2	160
3	1280
4	10240
5	81920
6	655360
7	5242880

超时间隔逐次增大，比固定超时间隔的好处在于：

- 首次超时重传间隔小，有利于快速重传尾丢失的 TP Packet，有助于误码丢包等非拥塞场景造成的偶发性丢包进行快速恢复。
- 当网络出现拥塞时，会导致 RTT 变长。通过逐次增加超时重传时间，相比设置较小的超时时间，可以减少无效重传，避免网络带宽的浪费；相比设置较大的超时时间，可以减少尾包丢失时的等待时间。
- TPEP 感知的一条端到端的 TP Channel 在网络上存在多条可达路径。当多平面网络故障导致一条路径不可达时，可能会发生需要路由重新收敛的情况。而 TPEP 连续超时重传一定次数后，会判定传输路径不可达。当超时间隔设置较小时，可能导致 TPEP 判定传输路径不可达所经历的时间小于路由下发所需时间，从而误判传输路径不可达。

#### 6.4.2.2 GoBackN 重传

GoBackN 重传可以配合快速重传和超时重传使用。UB 传输层在启用 GoBackN 重传算法-配合快速重传时，重传由 TPNAK 触发，TP Sender 根据收到的错误应答 TPNAK，重传确认接收到的 TP Packet 的后续所有 TP Packet。TP Receiver 收到 TP Packet 时需根据其携带的 PSN 判断 TP Packet 所在的区间（见 6.4.1 节）并回复应答包 TPACK 或 TPNAK，其中 TPACK 为正确应答，TPNAK 为错误应答。TPACK 中会携带 PSN 信息，其标识了被 TP Receiver 正确并按序接收的 TP Packet 的最大序列号；TPNAK 中携带的 PSN 域段标识 EPSN 信息。TP Sender 通过 TP Receiver 的应答包确认 TP Packet 的接收状态，并根据 TPNAK 返回的 PSN 域段重传此 PSN 及其后续所有已发送的 TP Packet。UB 传输层在启用 GoBackN 重传算法-无快速重传时，重传完全由超时触发；TP Sender 在触发超时后，重传未被 TPACK 确认的所有 TP Packet。

GoBackN 算法实现简单、重传的数据量较大，一般用于丢包率非常低的场景。在逐包负载均衡场景下，一般不配合快速重传使用，以避免不必要的重传。

GoBackN 重传下，传输层应答类型由 RTPH 中的 TPOopcode 和 RSPST 域段联合指定：对于不携带拥塞控制扩展包头且非事务层 ROL 模式的应答包，TPACK 由 TPOopcode=0x2, RSPST=3'b000 域段表征，TPNAK 类型由 TPOopcode=0x2、RSPST=3'b011 表征。对于携带拥塞控制扩展包头或事务层 ROL 模式下的应答包域段设置参照 6.2.1 节相关要求。

#### 6.4.2.2.1 GoBackN 重传-配合快速重传

一般来说，在 TP Packet 不会因为路由产生乱序的场景下（如启用逐流负载均衡场景），GoBackN 重传可以配合快速重传使用。

TP Receiver 收到 TP Packet 时根据其携带的 PSN 与本地维护的 EPSN 判断 TP Packet 所在的区间（见 6.4.1 节），更新本地接收状态如 EPSN，并回应应答包将接收信息带回 TP Sender。(i) 如果 TP Packet 是正序包，TP Receiver 正常接收，TP Receiver 将 EPSN 的值增加 1，并回复 TPACK，其中 RTPH.PSN 值为 TP Receiver 维护的 EPSN-1。(ii) 如果 TP Receiver 接收到的 TP Packet 落在重复区间，丢弃 TP Packet，本地维护的 EPSN 值不变；回复 TPACK，其携带的 PSN 值为 EPSN-1。(iii) 如果 TP Receiver 接收到的 TP Packet 落在乱序区间，则 TP Receiver 丢弃乱序包；本地维护的 EPSN 值不变；回复错误应答 TPNAK，其携带的 PSN 值为 EPSN。特别地，如果连续收到乱序包，不重复回 TPNAK。(iv) 如果 TP Packet 落在无效区间，直接丢弃该 TP Packet，本地维护的 EPSN 值不变；不回应应答。

TP Sender 在收到的错误应答 TPNAK 时，立即重传 TPNAK 所确认接收到的 TP Packet 的后续所有 TP Packet。

本章节中结合典型场景描述 GoBackN 重传-配合快速重传算法，在 TP Packet 丢失和应答包丢失场景下的流程。其中 TP Packet 丢失，分为首次丢失、非首次丢失和尾丢失三种典型场景；应答包丢失分为 TPACK 丢失、TPACK 尾丢失和 TPNAK 丢失三种典型场景。

#### TP Packet 丢失重传流程

##### 1. 首次 TP Packet 丢失

GoBackN 重传-配合快速重传，在 TP Packet 首次丢失场景下的重传流程如图 6-18 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m)到达 TP Receiver。TP Receiver 根据收到的 TP Packet 的 PSN 判断该 TP Packet 是否为正序包。因为 TP Receiver 维护的 EPSN=m 与 TP Packet(PSN=m)携带的 PSN 相同，TP Receiver 判断 TP Packet(PSN=m)为正序包，正确接收 TP Packet(PSN=m)并回复 TPACK(PSN=m)后更新 EPSN 为 m+1。
- (3) TP Packet(PSN=m+1)在传输过程中被丢失。
- (4) TP Packet(PSN=m+2)到达 TP Receiver，因为此时 TP Receiver 维护的 EPSN 为 m+1，TP Receiver 判断 TP Packet(PSN=m+2)为乱序包，丢弃 TP Packet(PSN=m+2)，并且回复 TPNAK(PSN=m+1)，其中 PSN=m+1 标识 TP Receiver 期待接收到的 PSN。

- (5) TP Packet(PSN=m+3)到达 TP Receiver，因为此时 TP Receiver 维护的 EPSN 为 m+1，TP Receiver 判断 TP Packet(PSN=m+3)为乱序包，丢弃 TP Packet(PSN=m+3)。因为已经回复过 TPNAK(PSN=m+1)，不重复回复应答包。
- (6) TP Sender 收到 TPNAK(PSN=m+1)后开始重传 PSN=m+1 之后的所有 TP Packet，即从 PSN=m+1 至 PSN=m+3 的 TP Packet。
- (7) TP Receiver 正确接收到 PSN=m+1 至 PSN=m+3 的 TP Packet 后分别回复 PSN=m+1 至 PSN=m+3 的 TPACK 并被 TP Sender 正确接收。

此外，为了减少 TPACK 回复的开销，TP Receiver 可以支持聚合 TPACK，即成功接收多个 TP Packet 后回复 TPACK。

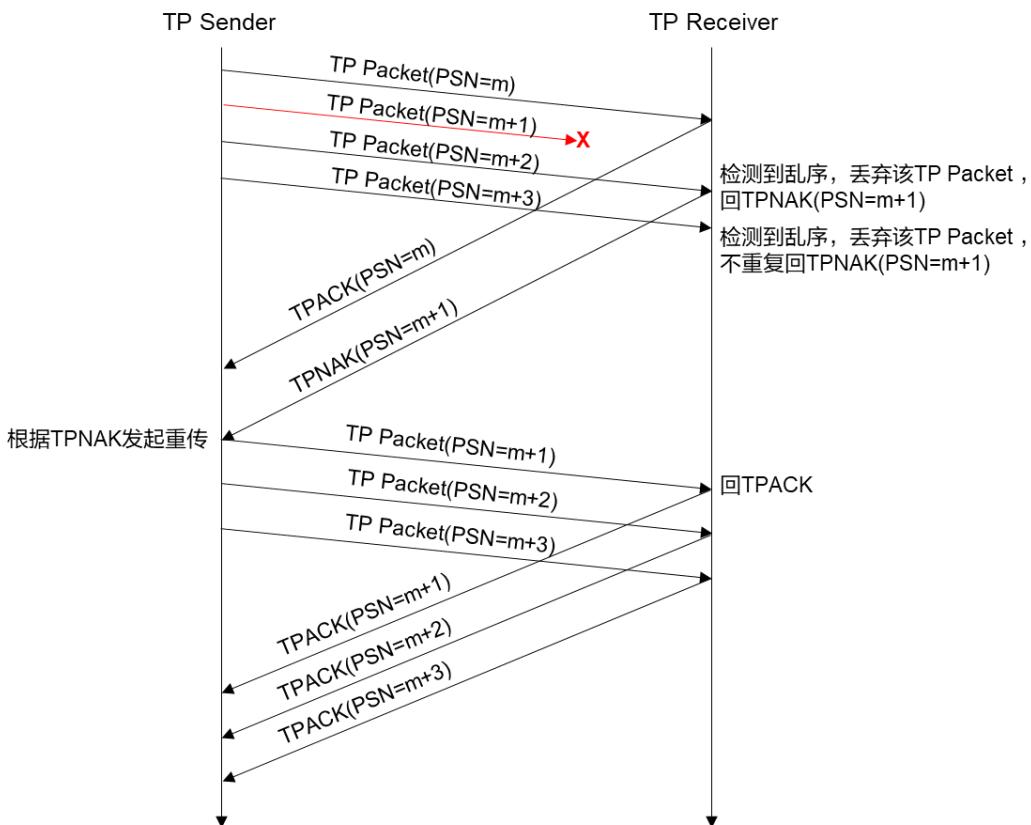


图 6-18 GoBackN 重传-配合快速重传，首次 TP Packet 丢失传输流程

## 2. 非首次 TP Packet 丢失

GoBackN 重传-配合快速重传，在 TP Packet 非首次丢失场景下的重传流程如图 6-19 所示。

- (1) TP Packet(PSN=m+1)首次丢失后发起重传的逻辑和首次丢失重传(1)-(6)流程相同，此处不赘述。
- (2) TP Packet(PSN=m+1)在传输过程中被再次丢失。
- (3) TP Receiver 收到重传的 TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)。因为此时 TP Receiver 维护的 EPSN 为 m+1，TP Receiver 判断 TP Packet(PSN=m+2)和 TP

Packet(PSN=m+3)为乱序包，丢弃 packet；而因为 TP Receiver 已经回复过 TPNAK(PSN=m+1)，TP Receiver 不重复回应应答包。

- (4) TP Sender 等待 TPACK 的时间达到 RTO, 触发超时重传, 开始重传 PSN=m+1 至 PSN=m+3 的 TP Packet。
- (5) TP Receiver 正确接收到 PSN=m+1 至 PSN=m+3 的 TP Packet 后回复 PSN=m+1 至 PSN=m+3 的 TPACK 并被 TP Sender 正确接收。

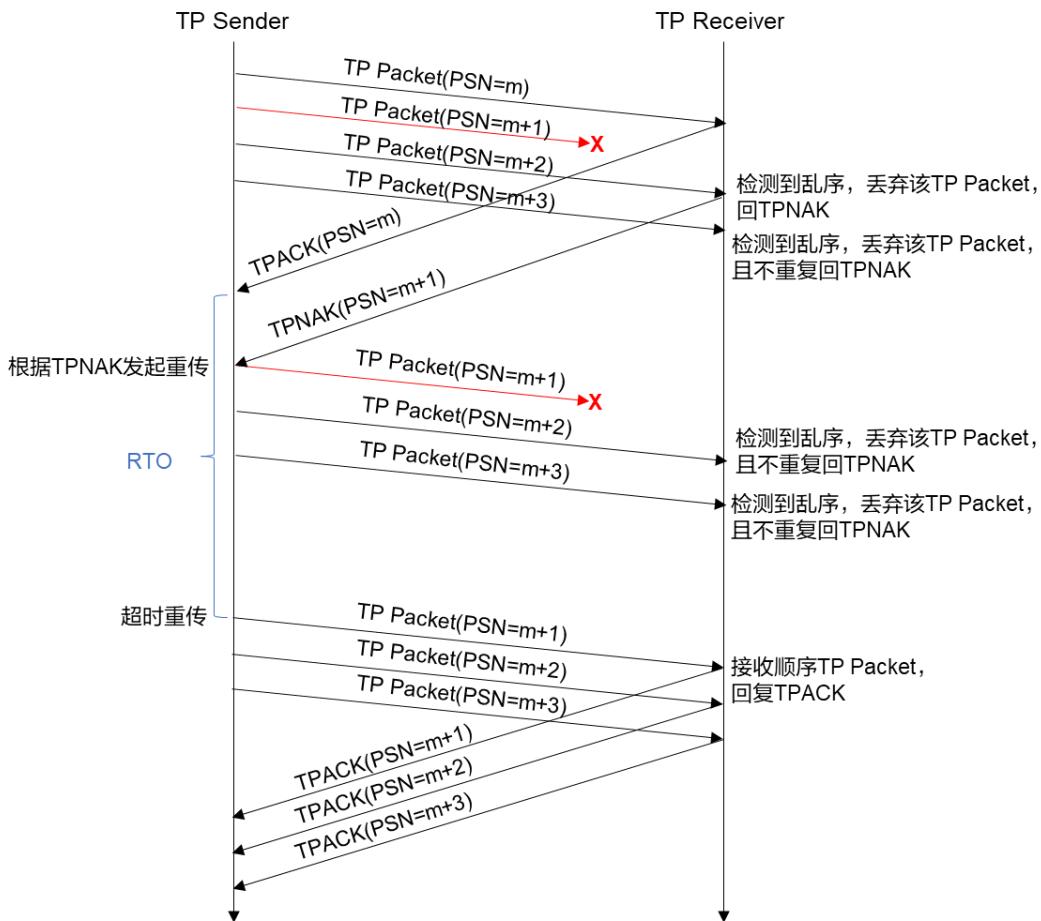


图 6-19 GoBackN 重传-配合快速重传, 非首次 TP Packet 丢失传输流程

### 3. 尾 TP Packet 丢失

GoBackN 重传-配合快速重传, 在 TP Packet 尾丢失场景下的重传流程如图 6-20 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m)和 TP Packet(PSN=m+1)被 TP Receiver 正确接收, TP Receiver 回复应答包 TPACK(PSN=m)和 TPACK(PSN=m+1)并被 TP Sender 正确接收。
- (3) TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)在传输过程中被丢失。
- (4) TP Sender 等待 TPACK 的时间达到 RTO, 触发超时重传, 开始重传 PSN=m+2 和 PSN=m+3 的 TP Packet。

- (5) TP Receiver 正确接收到 PSN=m+2 和 PSN=m+3 的 TP Packet 后分别回复 PSN=m+2 和 PSN=m+3 的 TPACK 并被 TP Sender 正确接收。

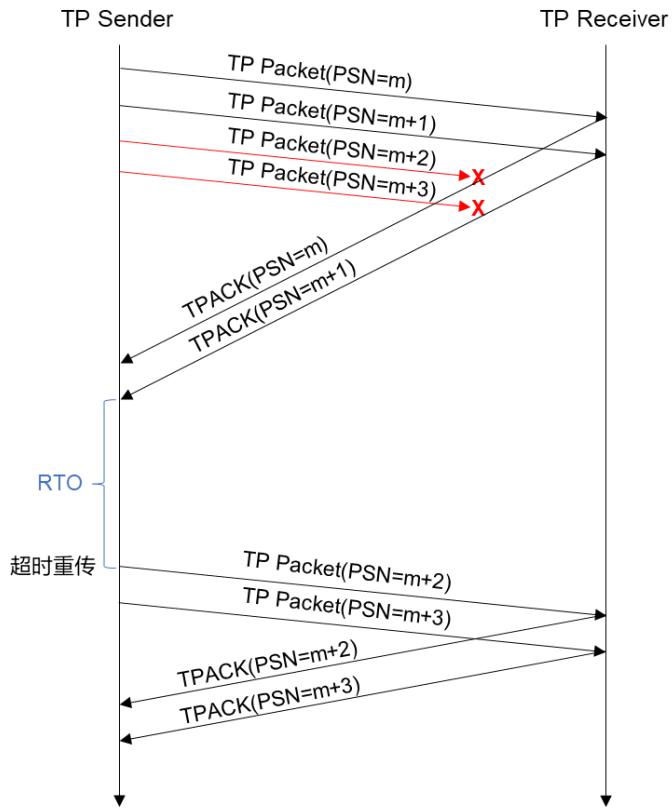


图 6-20 GoBackN 重传-配合快速重传，尾 TP Packet 丢失传输流程

### TPACK/TPNAK 丢失重传流程

#### 1. TPACK 丢失，但后续 TPACK 未丢失

GoBackN 重传-配合快速重传，在 TPACK 丢失场景下的重传流程如图 6-21 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+2 的 TP Packet 并被 TP Receiver 正确接收。
- (2) TP Receiver 回复 PSN=m 至 PSN=m+2 的 TPACK, TPACK(PSN=m) 和 TPACK(PSN=m+2) 被 TP Sender 正确接收，TPACK(PSN=m+1) 在传输过程中被丢失。
- (3) TP Sender 收到 TPACK(PSN=m+2)，因为 TPACK(PSN=m+2) 可以累计确认 PSN<=m+2 的 TP Packet 被正确接收，所以 TP Sender 会更新发送状态为 PSN<=m+2 的 TP Packet 均被正确接收。

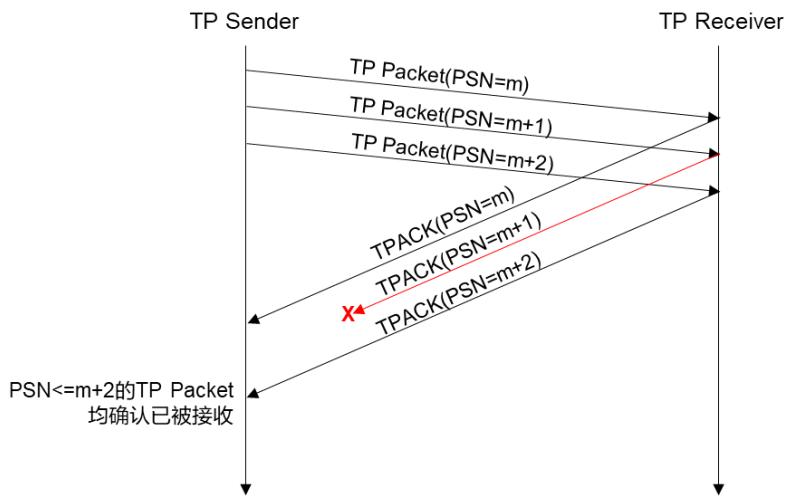


图 6-21 GoBackN 重传-配合快速重传，TPACK 丢失传输流程

## 2. 尾 TPACK 丢失

GoBackN 重传-配合快速重传，在 TPACK 尾丢失场景下的重传流程如图 6-22 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+2 的 TP Packet 并被 TP Receiver 正确接收。
- (2) TP Receiver 回复 PSN=m 至 PSN=m+2 的 TPACK, TPACK(PSN=m) 和 TPACK(PSN=m+1) 被 TP Sender 正确接收, TPACK(PSN=m+2) 在传输过程中被丢失。
- (3) TP Sender 等待 TPACK 的时间达到 RTO, 触发超时重传, 开始重传 TP Packet(PSN=m+2)。
- (4) TP Packet(PSN=m+2) 到达 TP Receiver, TP Receiver 检测到该 TP Packet 落在重复包区间, 丢弃该 TP Packet 并回复 TPACK(PSN=m+2)。
- (5) TPACK(PSN=m+2) 被 TP Sender 正确接收。

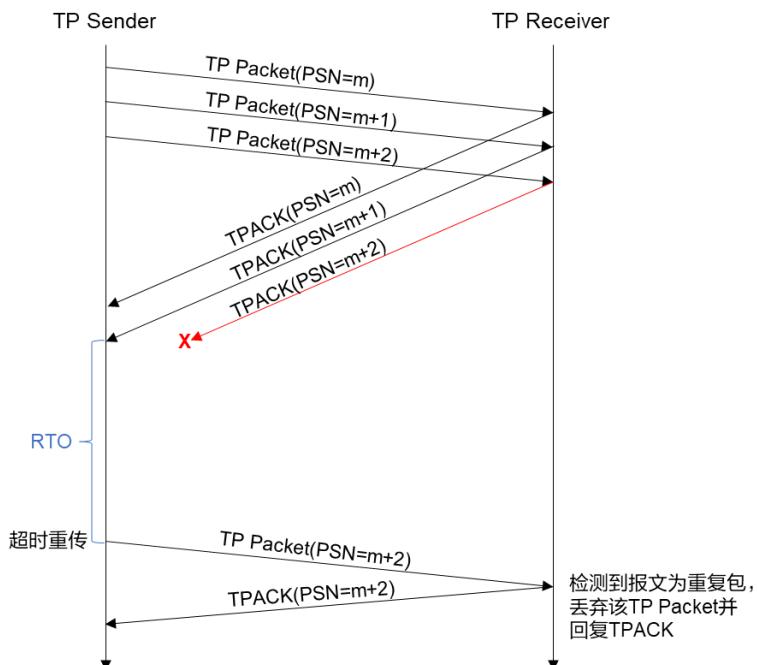


图 6-22 GoBackN 重传-配合快速重传，尾 TPACK 丢失传输流程

### 3. TPNAK 丢失

GoBackN 重传-配合快速重传，在 TPNAK 丢失场景下的重传流程如图 6-23 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+2 的 TP Packet。
- (2) TP Packet(PSN=m)被 TP Receiver 正确接收，TP Receiver 回复 TPACK(PSN=m)。
- (3) TP Packet(PSN=m+1)在传输过程中被丢失，乱序包 TP Packet(PSN=m+2)到达 TP Receiver 后被丢弃，TP Receiver 回复 TPNAK(PSN=m+1)。
- (4) TPNAK(PSN=m+1)在传输过程中被丢失。
- (5) TP Sender 等待 TPACK 的时间达到 RTO，触发超时重传，开始重传 TP Packet(PSN=m+1) 和 TP Packet(PSN=m+2)。
- (6) TP Packet(PSN=m+1)和 TP Packet(PSN=m+2)被 TP Receiver 正确接收后，TP Receiver 回复 TPACK(PSN=m+1)和 TPACK(PSN=m+2)并被 TP Sender 正确接收。

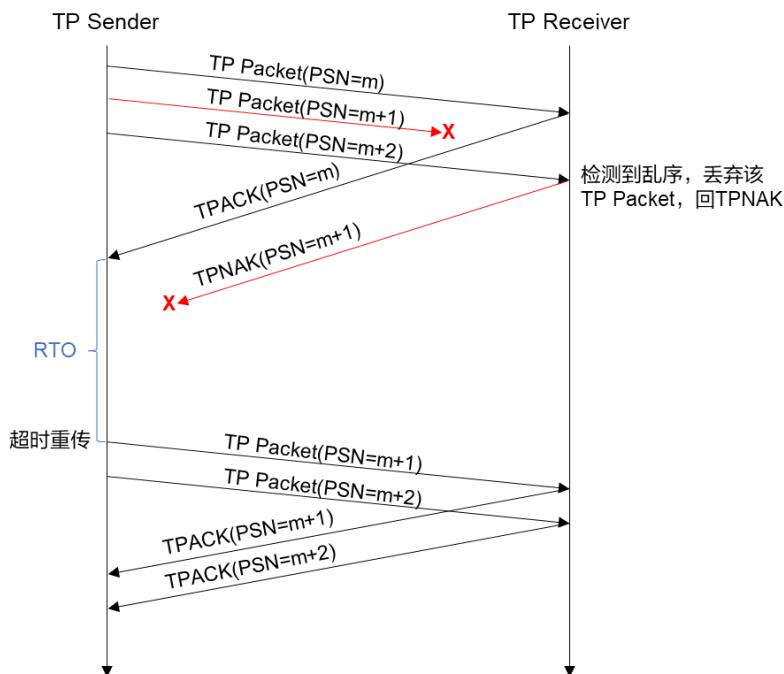


图 6-23 GoBackN 重传-配合快速重传，TPNAK 丢失传输流程

#### 6.4.2.2 GoBackN 重传-无快速重传

在 TP Packet 会因为路由产生乱序（如启用逐包负载均衡场景）的场景下，GoBackN 重传一般不启用快速重传，且此时需要使能 TP Receiver 的乱序接收能力，维护用于接收乱序包的 BitMap。在此类场景下，TP Receiver 接收到乱序包并不意味着一定发生了 TP Packet 丢失，所以使用超时重传减少或避免不必要的 TP Packet 重传。

TP Receiver 收到 TP Packet 时根据其携带的 PSN 与本地维护的 EPSN 判断 TP Packet 所在的区间（见 6.4.1 节），更新本地接收状态如 EPSN/BitMap，并回应应答包将接收信息带回 TP Sender。(i) 如果 TP Packet 是正序包，TP Receiver 正常接收，并更新本地用于乱序接收的 BitMap；本地维护的 EPSN 为

TP Receiver 接收到的最大正序 TP Packet PSN 加 1；TP Receiver 回复 TPACK，其中 RTPH.PSN 域段携带的 PSN 值为 EPSN-1。(ii) 如果 TP Receiver 接收到的 TP Packet 落在重复区间，则丢弃 TP Packet，EPSN 值不变；回复 TPACK，其携带的 PSN 值为 EPSN-1。(iii) 如果 TP Receiver 接收到的 TP Packet 落在乱序区间，若本地用于乱序接收的 BitMap 指示数据包未被接收过，则 TP Receiver 接收此乱序包，并更新 BitMap；若 BitMap 指示已接收过此数据包，则丢弃此数据包；EPSN 值不变；不回应应答。(iv) 如果 TP Packet 落在无效区间，直接丢弃该 TP Packet，EPSN 值不变；不回应应答。

TP Sender 需要根据收到的 TPACK 更新发送状态，维护收到的 TPACK 中携带的最大 PSN（记为 MaxPSN）；并且在 TP Packet 发出后启动超时重传的定时器，在超时发生时，重传所有  $\text{PSN} >= \text{MaxPSN} + 1$  的 TP Packet。

本章节中结合典型场景描述 GoBackN 重传-无快速重传，在 TP Receiver 接收到属于乱序区间的 TP Packet 和应答包丢失场景下的流程。其中 TP Receiver 接收到属于乱序区间的 TP Packet 的场景，分为首次检测到 TP Packet 乱序（丢包未实际发生、或丢包实际发生）、TP Packet 非首次丢失、和尾丢失三种典型场景；应答包丢失分为 TPACK 丢失和 TPACK 尾丢失两种典型场景。

### TP Receiver 接收到乱序区间 TP Packet

#### 1. 首次检测到 TP Packet 乱序（丢包未实际发生）

GoBackN 重传-无快速重传，在首次检测到 TP Packet 乱序（丢包未实际发生），即 TP Receiver 接收到属于乱序区间的 TP Packet 场景下的重传流程如图 6-24 所示。

- (1) TP Sender 发送  $\text{PSN}=m$  至  $\text{PSN}=m+3$  的 TP Packet。
- (2) TP Packet( $\text{PSN}=m$ )被 TP Receiver 正确接收，TP Receiver 回复 TPACK( $\text{PSN}=m$ )，TP Receiver 更新 EPSN 为  $m+1$ 。
- (3) TP Packet( $\text{PSN}=m+2$ ) 和 TP Packet( $\text{PSN}=m+3$ ) 到达 TP Receiver，此时 TP Packet( $\text{PSN}=m+1$ )还未收到。因为 TP Receiver 使能乱序接收能力，乱序包 TP Packet( $\text{PSN}=m+2$ )和 TP Packet( $\text{PSN}=m+3$ )被 TP Receiver 正确接收。注意因为此时重传触发的条件是超时，TP Receiver 无需回复 TPNAK。
- (4) TP Packet( $\text{PSN}=m+1$ )在传输过程中因为网络拥塞等原因晚于 TP Packet( $\text{PSN}=m+2$ )和 TP Packet( $\text{PSN}=m+3$ )到达 TP Receiver，TP Packet( $\text{PSN}=m+1$ )携带的 PSN 和 TP Receiver 维护的 EPSN 相同，即此时 TP Receiver 收到顺序包，回复 TPACK( $\text{PSN}=m+3$ )。
- (5) TP Sender 收到 TPACK( $\text{PSN}=m+3$ )，因为 TPACK( $\text{PSN}=m+3$ )可以累计确认  $\text{PSN} <= m+3$  的 TP Packet 被正确接收，所以 TP Sender 会更新发送状态为  $\text{PSN} <= m+3$  的 TP Packet 均被正确接收，TP Sender 取消超时重传计时。

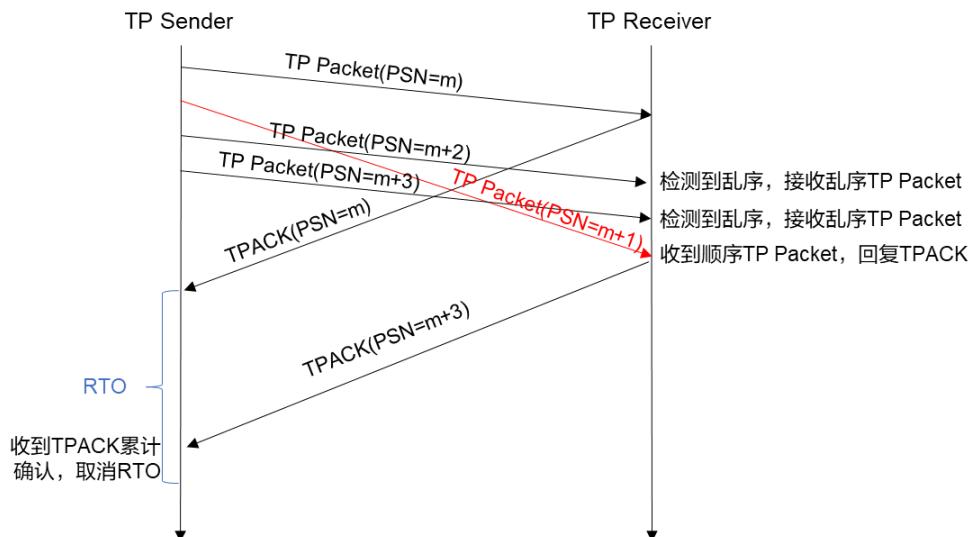
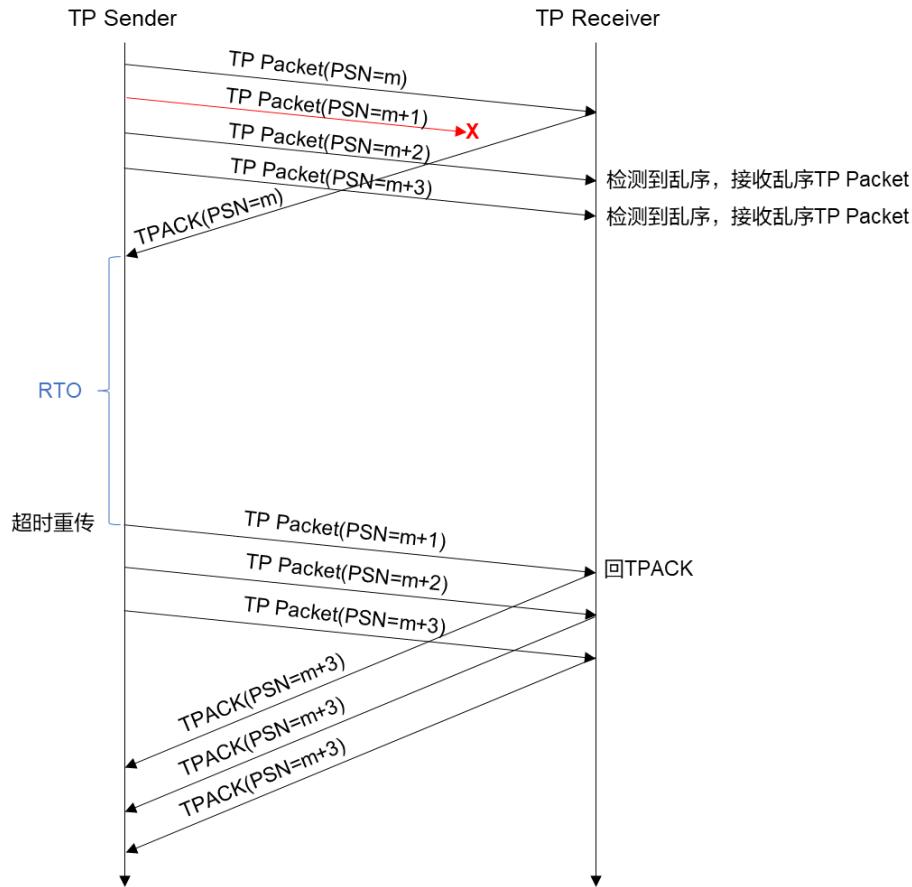


图 6-24 GoBackN 重传-无快速重传，首次检测到 TP Packet 乱序（丢包未实际发生）传输流程

## 2. 首次检测到 TP Packet 乱序（丢包实际发生）

GoBackN 重传-无快速重传，在首次检测到 TP Packet 乱序（丢包实际发生）场景下的重传流程如图 6-25 所示。需要指出的是，这种情况在链路质量较高、丢包率非常低的场景下出现的可能性较低。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m)被 TP Receiver 正确接收，TP Receiver 回复 TPACK(PSN=m)。
- (3) TP Packet(PSN=m+1)在传输过程中被丢失。
- (4) TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)到达 TP Receiver，因为 TP Receiver 使能乱序接收能力，乱序包 TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)被 TP Receiver 正确接收。
- (5) TP Sender 等待 TPACK 的时间达到 RTO, 触发超时重传, 开始重传 PSN=m+1 至 PSN=m+3 的 TP Packet。
- (6) TP Receiver 正确接收 TP Packet(PSN=m+1)并回复 TPACK(PSN=m+3); TP Receiver 检测 TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)为重复包, 丢弃并分别回复 TPACK(PSN=m+3)。



**图 6-25 GoBackN 重传-无快速重传，首次检测到 TP Packet 乱序（丢包实际发生）传输流程**

### 3. 非首次 TP Packet 丢失

GoBackN 重传-无快速重传时，非首次 TP Packet 丢失的重传流程与首次检测到 TP Packet 乱序（丢包实际发生）的重传流程类似。

### 4. 尾 TP Packet 丢失

GoBackN 重传-无快速重传时，尾 TP Packet 丢失的重传流程与 GoBackN 重传-配合快速重传时，尾 TP Packet 丢失的重传流程类似；因为在尾包丢失场景下，此两种重传算法触发重传的条件均为超时。

#### TPACK 丢失

GoBackN 重传-无快速重传时，TPACK 丢失的重传流程与 GoBackN 重传-配合快速重传时，TPACK 丢失的重传流程类似；因为在 TPACK 丢失场景下，此两种重传算法触发重传的条件均为超时。

#### 6.4.2.3 选择性重传

在 UB 传输层启用选择性重传时，TP Sender 根据 TPSACK 携带的 TP Packet 接收情况，只重传 TP Receiver 未收到的 TP Packet，不重传已经被 TP Receiver 正确接收的 TP Packet。选择性重传机制相比 GoBackN 重传，重传的数据量小，但实现相对复杂，且需要 TP Receiver 支持乱序接收，即 TP Receiver

可以正确接收一定乱序范围内的乱序包（见 6.5.1.4.1 节）。选择性重传一般用于链路质量较低、丢包率相对较高的场景。

启用选择性重传时，TP Receiver 本地维护 BitMap，记录接收到的 TP Packet 的 PSN。TP Receiver 收到 TP Packet 时需根据 TP Packet 携带的 PSN 判断其所在的区间（见 6.4.1 节）并回复应答包 TPACK 或 TPSACK，其中 TPACK 为正确应答，TPSACK 为选择性应答。TP Receiver 回复的应答包中会携带 PSN 信息，其标识了被 TP Receiver 正确并按序接收的 TP Packet 的最大序列号；TPSACK 中还额外携带了标识 TP Packet 详细接收情况的 SAETPH.BitMap，表征序列号在[RTPH.PSN, RTPH.PSN+SAETPH.BitMapSize-1]范围内的 TP Packet 接收情况。TP Sender 通过 TP Receiver 的应答包确认 TP Packet 的发送状态，并根据 TPSACK 返回的 BitMap 域段重传未被确认接收的 TP Packet。

TP Receiver 收到 TP Packet 时需根据其携带的 PSN 判断其所在的区间，并发送 TPACK/TPSACK 应答包，是否发送 TPSACK 取决于 TP Receiver 维护的 BitMap 范围内是否存在乱序包。(i) 如果 TP Packet 是正序包，则 TP Receiver 正常接收，EPSN 更新为接收到的最大正序包的 PSN+1；若此时 TP Receiver 的 BitMap 中没有乱序包，回复 TPACK，其中 RTPH.PSN 域段携带的 PSN 值为 EPSN-1；若此时 TP Receiver 的 BitMap 中存在乱序包，回复 TPSACK，其中 RTPH.PSN 域段携带的 PSN 值与 BitMap[0]所表征的 PSN 相同，TP Sender 根据 BitMap 中第一个 0 值的前一个 PSN 确定被顺序接收的最大 PSN。(ii) 如果 TP Receiver 接收到的 TP Packet 落在重复区间，丢弃 TP Packet，EPSN 值不变；类似地，依据接收端 TP Receiver 的 BitMap 中是否存在乱序包，回复 TPACK/TPSACK。(iii) 如果 TP Receiver 接收到的 TP Packet 落在乱序区间，EPSN 值不变；回复 TPSACK。(iv) 如果 TP Packet 落在无效区间，直接丢弃 TP Packet，EPSN 值不变；不回复应答。

TP Sender 接收到 TPSACK 时，若 MaxRcvPSN ( TP Receiver 收到的最大 PSN ) < BitMap 所能表达的最大 PSN，即 RTPH.PSN + BitMapSize ( 其中 BitMapSize 表征标识的 bit 数 )，则重传 PSN 区间 [RTPH.PSN+1, MaxRcvPSN] 中 TP Receiver 未收到的 TP Packet；否则重传 PSN 区间 [RTPH.PSN+1, RTPH.PSN+BitMapSize] 中 TP Receiver 未收到的 TP Packet。上述设计考虑 MaxRcvPSN 可能大于等于 TPSACK 包 BitMap 所能代表的最大 PSN 的情况。例如，TP Receiver 收到乱序包，但申请不到 BitMap，TP Receiver 丢弃此乱序包，但 MaxRcvPSN 仍更新为此 PSN。MaxRcvPSN 单调递增。

选择性重传可以配合快速重传和超时重传使用，两者的重传触发条件不同：在启用快速重传时，TP Sender 在收到 TPSACK 时更新 TP Packet 被接收的状态并立即开始重传；在不启用快速重传时，TP Sender 在收到 TPSACK 时仅做 TP Packet 接收状态的更新，等待超时后触发重传。

TP Receiver 发送的应答包类型由 RTPH 的 TPOopcode 和 RSPST 域段联合指定：对于不携带拥塞控制扩展包头且非事务层 ROL 模式的应答包，TPACK 由 TPOopcode=0x2，RSPST=3'b000 域段表征，TPSACK 类型由 TPOopcode=0x5 表征。对于 TPSACK 类型，需携带 SAETPH 包头，用于携带指示 TP Receiver 接收 TP Packet 情况的 BitMap 域段和标识 TP Receiver 收到的最大 PSN 的 MaxRcvPSN 域段，SAETPH 包头应符合 6.2.3 节相关要求。

#### 6.4.2.3.1 选择性重传-配合快速重传

一般来说，在 TP Packet 不会因为路由产生乱序的场景下（如启用逐流负载均衡场景），选择性重传可以配合快速重传使用。

本章节中结合典型场景描述选择性重传-配合快速重传算法，在 TP Packet 丢失或应答包丢失场景下的流程。其中 TP Packet 丢失，分为首次丢失、非首次丢失、和尾丢失三种典型场景；应答包丢失区分 TPACK 丢失或 TPACK 尾丢失两种典型场景。其中非首次丢失又分为不使用 MarkPSN 机制的场景和使用 MarkPSN 机制的场景。

##### MarkPSN 机制

为了避免重复重传，非首次丢失的 TP Packet 重传一般需要等待超时之后再次重传，而超时重传时间较难设置（见 6.4.2.1 节）。为此，UB 协议支持在逐流负载均衡模式下启用 MarkPSN 机制，优化非首次丢失的 TP Packet 的重传性能，使非首次丢失的 TP Packet 可以被快速识别并再次重传，无需等待超时重传。

MarkPSN 机制周期性地快速识别和重传 TP Packet。MarkPSN 机制将整个传输过程分为(i)新包发送阶段和(ii)丢包重传阶段，两个阶段交替进行。在新包发送阶段下，只发送新的 TP Packet，对于检测为丢失的 TP Packet，加入待重传集合而不进行重传；丢包重传阶段只重传上一新包发送阶段下记录的丢失的 TP Packet，不发送新的 TP Packet。两种阶段转移的关键变量为 MarkPSN。MarkPSN 由 TP Sender 为每个 TP Channel 单独维护，表示上一次丢包重传阶段转移后发出的第一个新 TP Packet 的 PSN，初始设置为首个 TP Packet 的 PSN。当处于新包发送阶段时，若 TP Sender 收到的 TPSACK 应答中指示大于等于 MarkPSN 序列号的 TP Packet 被接收，说明 MarkPSN 及以前的所有 TP Packet 已经被 TP Receiver 成功接收或已丢失，则新包发送阶段切换至丢包重传阶段。当处于丢包重传阶段时，若所有待重传的 TP Packet 均重传完毕，则切换回新包发送阶段，并用新发的第一个 TP Packet 的 PSN 更新 MarkPSN。

新包发送阶段需要准确检测丢失的 TP Packet。MarkPSN 机制为每个 TP Channel 记录最晚发出的首次重传 TP Packet 的 PSN(记为 LastFirstRtx)，用以区分首次丢失 TP Packet 和非首次丢失的 TP Packet。TPSACK 应答中指示的未接收到的 TP Packet 中，PSN 大于 LastFirstRtx 的 TP Packet 为首次丢失的 TP Packet，首次丢失的 TP Packet 立即被加入待重传集合；对于其它未接收到的 TP Packet，因为上一轮已经重传过，可能仍然在飞行中，因而需等待 TP Sender 收到的 TPSACK 应答中指示大于等于 MarkPSN 序列号的 TP Packet 被接收（即满足新包发送阶段到丢包重传阶段的跳转条件），才可被确认为非首次丢失 TP Packet 并被加入待重传集合，同时 TP Sender 进入新一轮丢包重传阶段。

特别地，存在两种边界情况使得 MarkPSN 机制不生效，此时退化为超时重传：

1. 重传完成后无新 TP Packet 可发送；
2. TP Packet 尾包丢失。

## TP Packet 丢失重传流程

### 1. 首次 TP Packet 丢失

当 TP Sender 收到指示 TP Packet 首次丢失的 TPSACK 应答时，会触发重传。为了避免重复重传，TP Sender 维护 HighRtxPSN，记录重传过的 TP Packet 的最大 PSN。收到新的 TPSACK 时，从 HighRtxPSN 开始重传。

选择性重传-配合快速重传，在 TP Packet 首次丢失场景下的重传流程如图 6-26 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m)到达 TP Receiver。TP Receiver 根据收到的 TP Packet 的 PSN 判断该 TP Packet 是否为正序包。因为 TP Receiver 维护的 EPSN=m 与 TP Packet(PSN=m)携带的 PSN 相同，TP Receiver 判断 TP Packet(PSN=m)为正序包，正确接收 TP Packet(PSN=m)并回复 TPACK(PSN=m)后更新 EPSN 为 m+1。
- (3) TP Packet(PSN=m+1)在传输过程中被丢失。
- (4) TP Packet(PSN=m+2)到达 TP Receiver，因为此时 TP Receiver 维护的 EPSN 为 m+1，TP Receiver 判断 TP Packet(PSN=m+2)为乱序包，接收 TP Packet(PSN=m+2)，更新维护的 Bitmap；并且回复 TPSACK(PSN=m,[1,0,1])，其中 PSN=m 标识 TP Receiver 顺序接收到的最大 PSN，bitMap[1,0,1]标识 TP Packet(PSN=m+1)未被正确接收，TP Packet(PSN=m)和 TP Packet(PSN=m+2)均已被正确接收。为了方便描述，本章节中使用 TPSACK(PSN,bitMap)的方式，在逻辑上表征 TPSACK 中 RTPH.PSN 和 Bitmap 所携带的信息。
- (5) TP Packet(PSN=m+3)到达 TP Receiver，类似地，TP Receiver 回复 TPSACK(PSN=m,[1,0,1,1])，标识 PSN=m+1 以外的其它 PSN<=m+3 的 TP Packet 均被正确接收。
- (6) TP Sender 收到 TPSACK(PSN=m,[1,0,1])，仅发起 TP Packet(PSN=m+1)的重传，并记录 HighRtxPSN=m+1。
- (7) TP Sender 收到 TPSACK(PSN=m,[1,0,1,1])，指示 TP Packet(PSN=m+1)未收到，但因为 TP Sender 维护的 HighRtxPSN=m+1 说明 TP Packet(PSN=m+1)通过 TPSACK 触发过重传，TP Sender 不进行重复重传。
- (8) TP Receiver 正确接收到 TP Packet(PSN=m+1)后回复 TPACK(PSN=m+3)并被 TP Sender 正确接收。

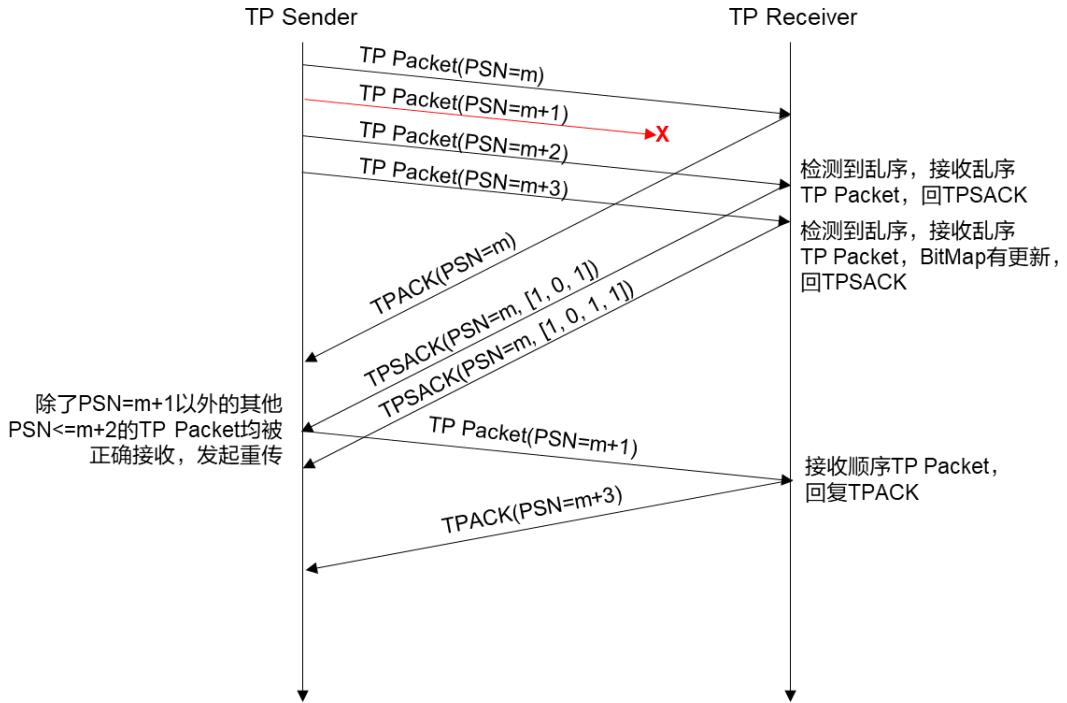


图 6-26 选择性重传-配合快速重传，首次 TP Packet 丢失传输流程

此外，为了减少 TPSACK 回复的开销，TP Receiver 可以支持聚合 TPSACK。

特别地，当因为 TP Receiver 申请不到 BitMap 导致乱序 TP Packet 被丢弃时，对应的 TPSACK 应答中不会携带此 TP Packet 的接收情况，也不会触发 TP Sender 对应 TP Packet 的重传。

## 2. 非首次 TP Packet 丢失（不使用 MarkPSN 机制）

选择性重传-配合快速重传，在 TP Packet 出现非首次丢失，且不使用 MarkPSN 机制场景下的重传流程如图 6-27 所示。

- (1) TP Packet(PSN=m+1)首次丢失后发起重传的逻辑和首次丢失重传(1)-(5)流程相同，此处不赘述。
- (2) TP Sender 发起 TP Packet(PSN=m+1)的重传，和新包 TP Packet(PSN=m+4)的传输。
- (3) TP Packet(PSN=m+1)在传输过程中被再次丢失。
- (4) TP Packet(PSN=m+4)到达 TP Receiver, TP Receiver 回复 TPSACK(PSN=m, [1,0,1,1,1])。
- (5) TP Sender 收到 TPSACK(PSN=m, [1,0,1,1,1])，因为无法区分此 TPSACK 是否为重传 TP Packet(PSN=m+1)到达前发出的应答包，因此不发起 TP Packet(PSN=m+1)的二次重传。
- (6) TP Sender 等待 TPACK 的时间达到 RTO，触发超时重传，重传 TP Packet(PSN=m+1)。
- (7) TP Receiver 正确接收到 TP Packet(PSN=m+1)，回复 TPACK(PSN=m+4)并被 TP Sender 正确接收。

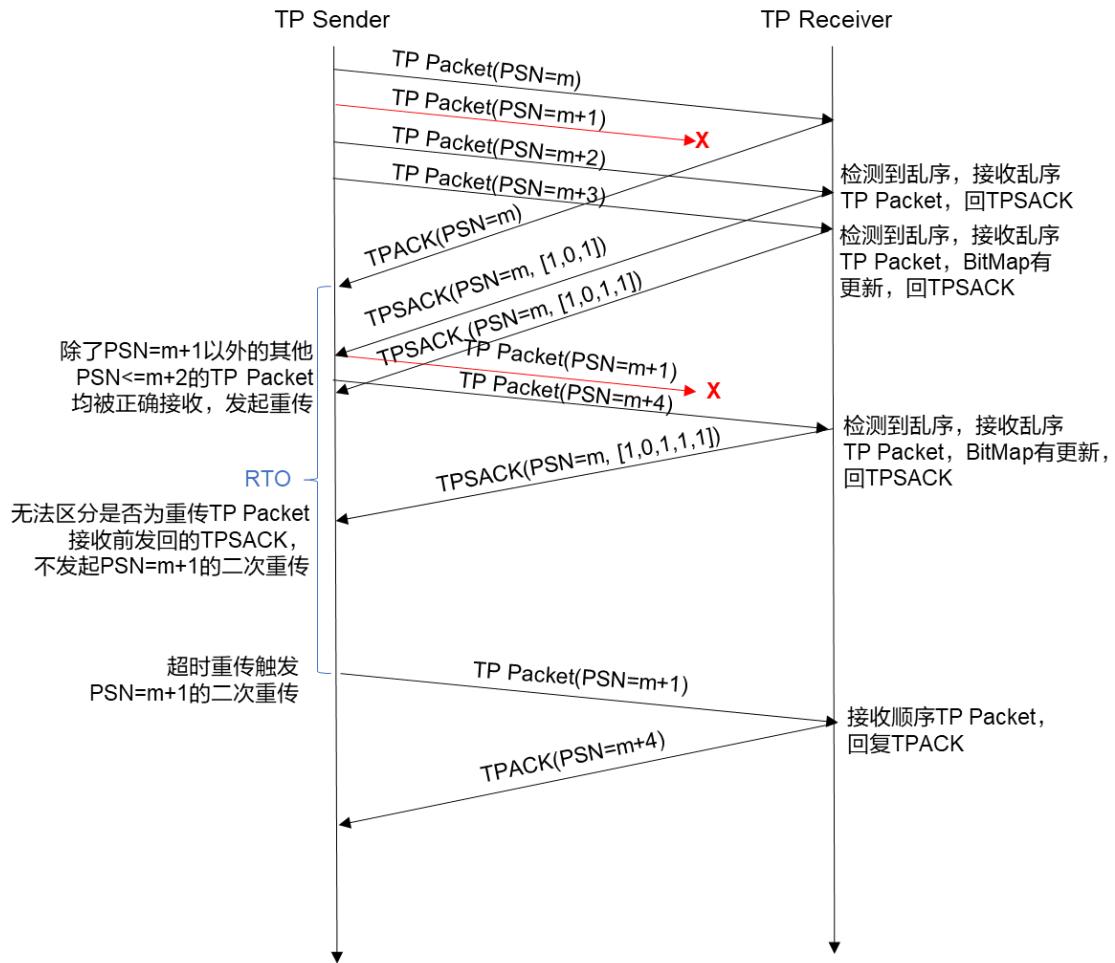


图 6-27 选择性重传-配合快速重传, 非首次 TP Packet (无 MarkPSN 机制) 丢失传输流程

### 3. 非首次 TP Packet 丢失 (使用 MarkPSN 机制)

选择性重传-配合快速重传, 在 TP Packet 非首次丢失, 且使用 MarkPSN 机制场景下的重传流程如图 6-28 所示。

- (1) TP Packet(PSN=m+1)首次丢失后发起重传的逻辑和首次丢失重传(1)-(5)流程相同, 此处不赘述。
- (2) TP Sender 发起 TP Packet(PSN=m+1)的重传, 和新包 TP Packet(PSN=m+4)的传输。因为 TP Packet(PSN=m+4)为重传 TP Packet 发送后发出的第一个 TP Packet, TP Sender 需要记录 MarkPSN=m+4。
- (3) TP Packet(PSN=m+1)在传输过程中被再次丢失。
- (4) TP Packet(PSN=m+4)到达 TP Receiver, TP Receiver 回复 TPSACK(PSN=m,[1,0,1,1,1])。
- (5) TP Sender 收到 TPSACK(PSN=m,[1,0,1,1,1]), 确认了除 PSN=m+1 以外的 PSN<=m+4 的 TP Packet 均被接收。因为此时 TP Sender 维护的 MarkPSN=m+4, 且 TP Packet(PSN=m+4) 被正确接收, 此时可以重传 TP Packet(PSN=m+1)。

- (6) TP Receiver 正确接收到 Packet(PSN=m+1)后回复 TPACK(PSN=m+4)并被 TP Sender 正确接收。

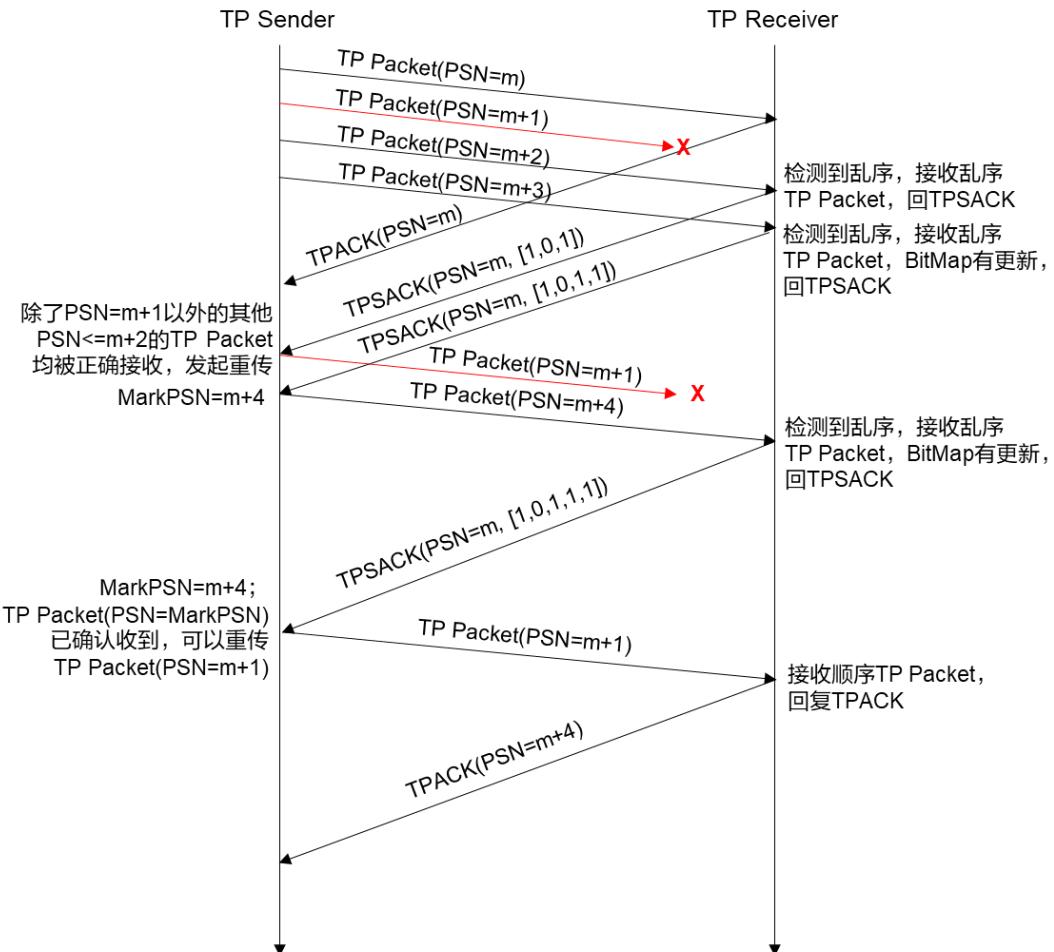


图 6-28 选择性重传-配合快速重传, 非首次 TP Packet ( 使用 MarkPSN 机制 ) 丢失传输流程

#### 4. 尾 TP Packet 丢失

选择性重传-配合快速重传, 在 TP Packet 尾丢失场景下的重传流程如图 6-29 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+4 的 TP Packet。
- (2) TP Packet(PSN=m+1)和 TP Packet(PSN=m+4)在传输过程中被丢失。
- (3) 与首次 TP Packet 场景中的流程类似, TP Sender 在收到 TPSACK(PSN=m, [1,0,1])后开始重传 TP Packet(PSN=m+1)并被 TP Receiver 成功接收。
- (4) 因为 TP Receiver 无法检测尾包丢失, TP Sender 等待 TPACK 的时间达到 RTO, 触发超时重传, 重传 TP Packet(PSN=m+4)。

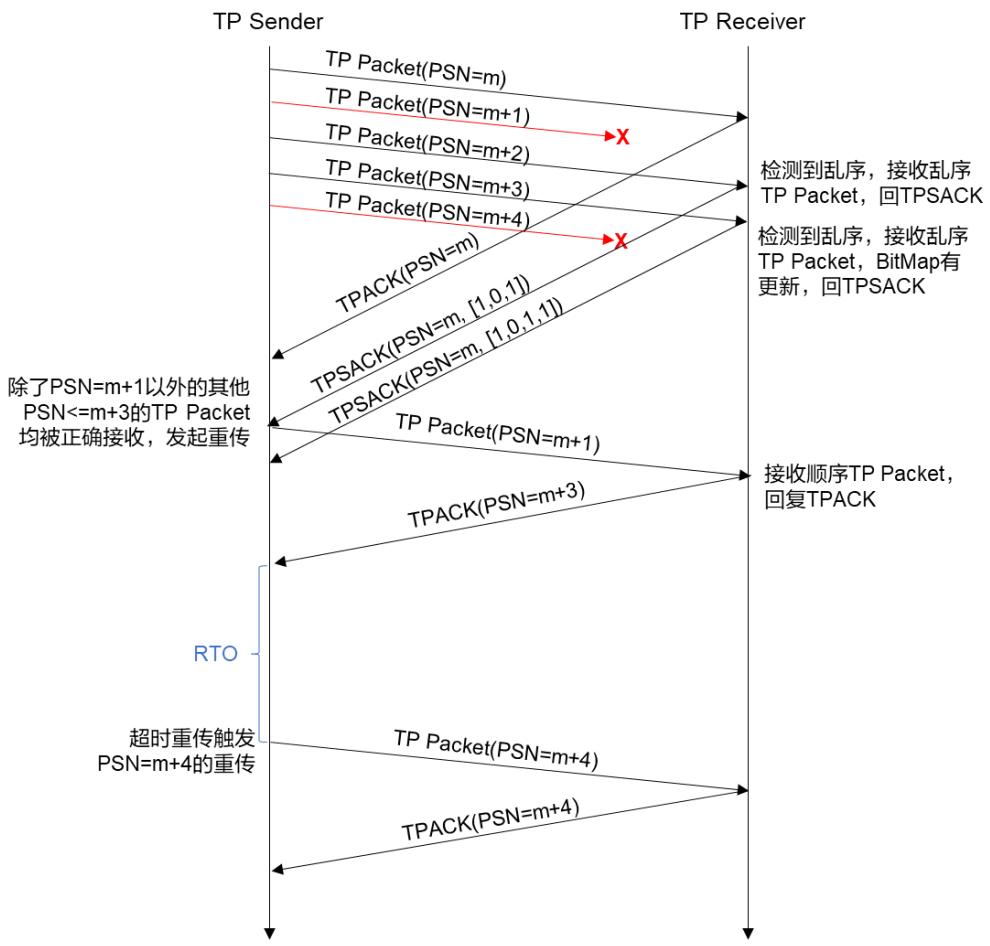


图 6-29 选择性重传-配合快速重传, 尾 TP Packet 丢失传输流程

### TPACK/TPSACK 丢失重传流程

#### 1. TPACK 丢失

选择性重传-配合快速重传时, TPACK 丢失的重传流程与 GoBackN 重传-配合快速重传时, TPACK 丢失的重传流程类似。

#### 2. 尾 TPACK 丢失

选择性重传-配合快速重传时, 尾 TPACK 丢失的重传流程与 GoBackN 重传-配合快速重传时, 尾 TPACK 丢失的重传流程类似。

#### 3. TPSACK 丢失

选择性重传-配合快速重传, 在 TPSACK 丢失场景下的重传流程如图 6-30 所示。

(1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。

(2) TP Packet(PSN=m+1)在传输过程中被丢失。

(3) TP Receiver 在收到 TP Packet(PSN=m+2)时回复 TPSACK(PSN=m,[1,0,1]); 类似地, 在收到 TP Packet(PSN=m+3)时回复 TPSACK(PSN=m,[1,0,1,1])。

(4) TPSACK(PSN=m,[1,0,1])在传输过程中被丢失。

- (5) TP Sender 在收到 TPSACK(PSN=m,[1,0,1,1]) 后确认 PSN=m+1 以外的其它 PSN<=m+3 的 TP Packet 均被正确接收，发起 TP Packet(PSN=m+1)的重传。
- (6) TP Receiver 正确接收到 Packet(PSN=m+1)后回复 TPACK(PSN=m+3)并被 TP Sender 正确接收。

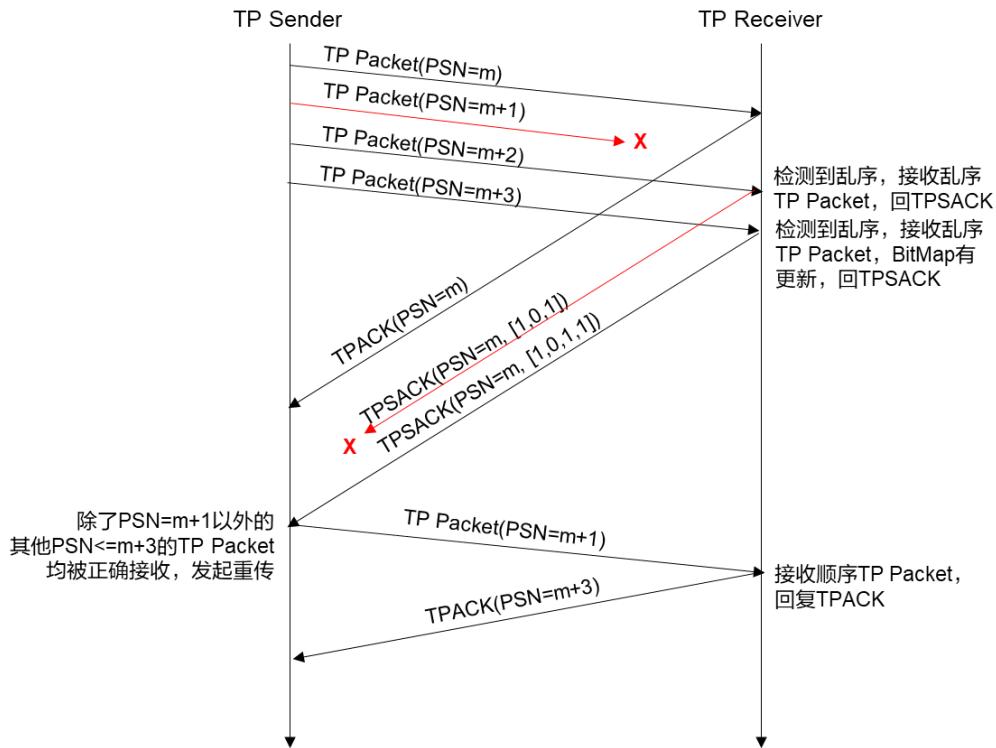


图 6-30 选择性重传-配合快速重传，TPSACK 丢失传输流程

#### 4. 尾 TPSACK 丢失

选择性重传-配合快速重传时，尾 TPSACK 丢失的重传流程与 GoBackN 重传-配合快速重传时，TPACK 丢失的重传流程类似；因为在尾包丢失场景下，此两种重传算法触发重传的条件均为超时。

##### 6.4.2.3.2 选择性重传-无快速重传

在 TP Packet 会因为路由产生乱序（如启用逐包负载均衡场景）的场景下，选择性重传一般不启用快速重传，且此时需要使能 TP Receiver 的乱序接收能力。在此类场景下，TP Receiver 接收到乱序 TP Packet 并不意味着一定发生了 TP Packet 丢失，用户可以配置 TP Receiver 设置累计接收的乱序 TP Packet 数量作为返回 TPSACK 的触发条件，用于减少连续收到乱序 TP Packet 时 TPSACK 的发送开销。

本章节中结合典型场景描述选择性重传-无快速重传，在 TP Receiver 接收到属于乱序区间的 TP Packet 和应答包丢失场景下的流程。其中 TP Receiver 接收到属于乱序区间的 TP Packet 的场景，分为首次检测到 TP Packet 乱序（丢包未实际发生、或丢包实际发生）、TP Packet 非首次丢失、和尾丢失三种典型场景。

### TP Receiver 接收乱序区间 TP Packet

#### 1. 首次检测到 TP Packet 乱序 (丢包未实际发生)

选择性重传-无快速重传，在首次检测到 TP Packet 乱序（丢包未实际发生），即 TP Receiver 接收到属于乱序区间的 TP Packet 场景下的重传流程如图 6-31 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m+1)在传输过程中因为网络拥塞等原因晚于 TP Packet(PSN=m+2)和 TP Packet(PSN=m+3)到达 TP Receiver。
- (3) TP Receiver 在收到 TP Packet(PSN=m+2) 和 TP Packet(PSN=m+3) 时分别回复 TPSACK(PSN=m, [1,0,1])和 TPSACK(PSN=m, [1,0,1,1])。
- (4) TP Sender 在收到 TPSACK(PSN=m, [1,0,1]) 和 TPSACK(PSN=m, [1,0,1,1]) 后确认 PSN=m+1 以外的其它  $PSN \leq m+3$  的 TP Packet 均被正确接收。
- (5) TP Receiver 在收到 TP Packet(PSN=m+1)后回复 TPACK(PSN=m+3)。
- (6) TP Sender 收到 TPACK(PSN=m+3)，因为 TPACK(PSN=m+3)可以累计确认  $PSN \leq m+3$  的 TP Packet 被正确接收，所以 TP Sender 会更新发送状态为  $PSN \leq m+3$  的 TP Packet 均被正确接收，TP Sender 取消超时重传计时。

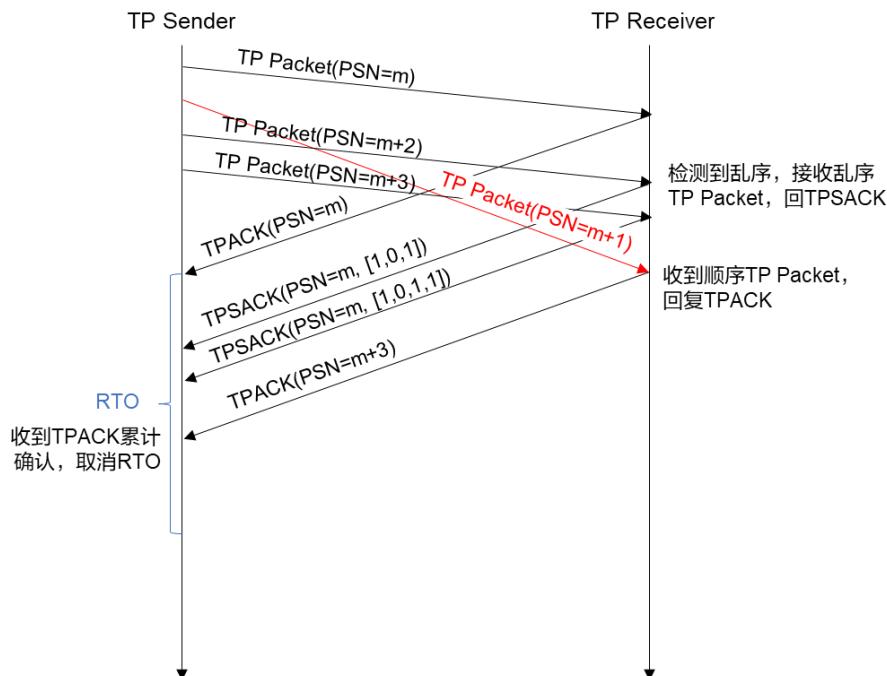


图 6-31 选择性重传-无快速重传，首次 TP Packet 丢失（丢包未实际发生）传输流程

#### 2. 首次 TP Packet 丢失（丢包实际发生）

选择性重传-无快速重传，在首次检测到 TP Packet 乱序（丢包实际发生），即 TP Receiver 接收到属于乱序区间的 TP Packet 场景下的重传流程如图 6-32 所示。

- (1) TP Sender 发送 PSN=m 至 PSN=m+3 的 TP Packet。
- (2) TP Packet(PSN=m+1)在传输过程中被丢失。

- (3) TP Receiver 在收到 TP Packet(PSN=m+2) 和 TP Packet(PSN=m+3) 时分别回复 TPSACK(PSN=m, [1,0,1]) 和 TPSACK(PSN=m, [1,0,1,1])。
- (4) TP Sender 等待 TPACK 的时间达到 RTO，触发超时重传，重传 TP Packet(PSN=m+1)。
- (5) TP Receiver 正确接收到 Packet(PSN=m+1) 后回复 TPACK(PSN=m+3) 并被 TP Sender 正确接收。

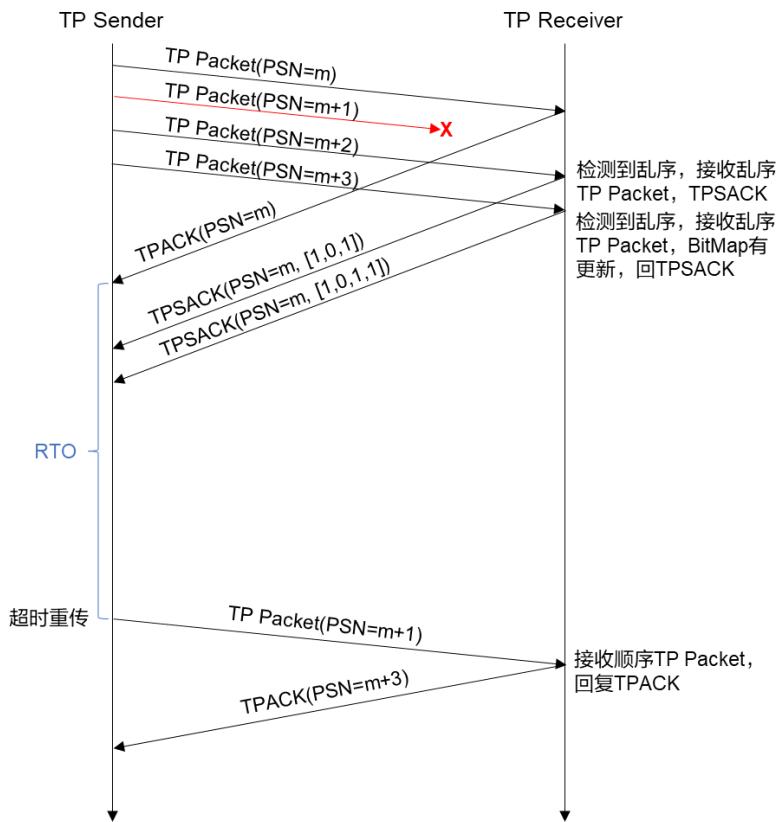


图 6-32 选择性重传-无快速重传，首次 TP Packet 丢失（丢包实际发生）传输流程

### 3. 非首次 TP Packet 丢失

选择性重传-无快速重传时，非首次 TP Packet 丢失的重传流程与选择性重传-配合快速重传时，非首次 TP Packet 丢失的重传流程类似。

### 4. 尾 TP Packet 丢失

选择性重传-无快速重传时，尾 TP Packet 丢失的重传流程与选择性重传-配合快速重传时，尾 TP Packet 丢失的重传流程类似。

## TPACK/TPSACK 丢失

选择性重传-无快速重传时，TPACK/TPSACK 丢失的重传流程与选择性重传-配合快速重传时，TPACK/TPSACK 丢失的重传流程类似，因为在 TPACK/TPSACK 丢失场景下，此两种重传算法触发重传的条件均为超时。

## 6.5 多路径负载均衡

### 6.5.1 RTP 多路径负载均衡

RTP 支持适配多路径负载均衡。为了充分利用 UBPU 和 UB Switch 的带宽，一般可以在一对 UBPU 之间建立多条 TP Channel，或启用逐包负载均衡机制，使得一对 UBPU 之间的流量使用多路径传输；且此两种机制可以叠加使用。其中逐包负载均衡机制可以通过 (i) 在 UB 传输层更换包头中 Srcport/LBF 等域段，从而主动控制 packet 在 UB Switch 上的路径选择的方式实现，也可以通过 (ii) 在 UB Switch 启用逐包负载均衡的方式实现（见 5.3.2 节，本章节不做赘述）。本章节中主要讨论在一对 UBPU 之间建立多条 TP Channel，以及利用方式(i)在传输层控制逐包负载均衡的两种机制。

在逐包负载均衡模式下，因为每条路径的带宽、时延、拥塞程度等可能不同，从而导致通过同一条 TP Channel 传输的 TP Packet 乱序到达传输层接收端。UB 传输层接收端支持乱序接收 TP Packet，并且支持配置相应的重传算法以避免或减少不必要的重传。

#### 6.5.1.1 TPG 机制

TPG 是 UB 传输层可靠传输服务为事务层提供传输服务的传输通道组。TPG 由若干个 TP Channel 构成，并对其 TP Channel 进行管理。TPG 负责将承载不同事务操作的 TP Packet 根据一定的策略（如 RoundRobin 地选择 TP Channel，或优先选择拥塞程度轻的 TP Channel）分发到不同的 TP Channel 上，实现流量在 TP Channel 间的负载均衡。TPG 需负责将相同的事务操作分发到一个相同的 TP Channel 进行发送。TPG 中的 TP Channel 可以绑定相同端口或不同端口，一个 TP Channel 只能隶属于一个 TPG。

示例：UB TPG 与事务层交互示例

TPG 的使用示意图如图 6-33 所示，图中每对 UBPU 间维护一个 TPG，每个 TPG 均包含 3 个 TP Channel，每个 TP Channel 两端为一对 TPEP。UBPU 之间进行数据传输时，事务层下发事务操作给传输层，TPG 负责将事务操作分发到其绑定的 TP Channel 中。

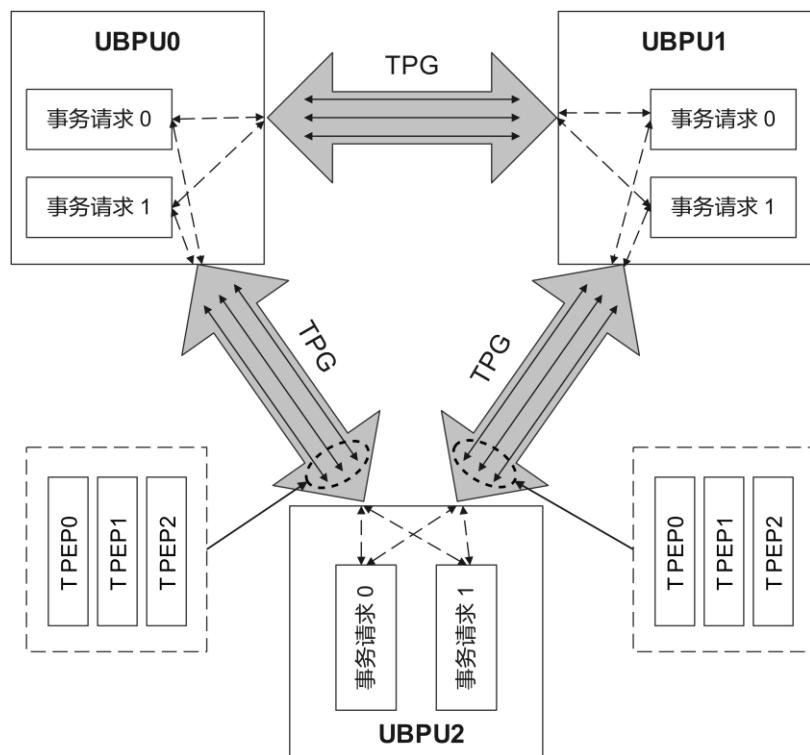


图 6-33 TPG 与事务层交互示意图

### 6.5.1.2 基于 TP Channel 负载均衡

一个 UBPU 可能包含多个端口，且一对 UBPU 之间可能存在多条网络路径，传输层可以通过在一对 UBPU 间建立多个 TP Channel，充分利用 UBPU 和 UB Switch 的带宽。通过 TPG 管理或者事务层选定 TP Channel 的方式，不同事务操作可以被调度到多个 TP Channel 中进行发送。

示例：TPG 多路径示例

UB 可以通过建立多 TP Channel 的方式在逐流负载均衡下使用网络多路径。表 6-9 描述了 TP Sender 建立多个 TP Channel 并由 TPG 管理，在使用逐流负载均衡、packet 使用 UDP 包格式时的示例（使用 CNA 包格式时，通过指定不同 LBF 字段建立不同 TP Channel，此处不赘述）：UBPU 发送端通过分别指定不同 UDP source port 对一个 UBPU 接收端建立了 TP Channel1、TP Channel2、TP Channel3、TP Channel4 共 4 个 TP Channel，并组成 TPG；其中 TP Channel1 和 TP Channel2 使用相同的源物理端口 1，TP Channel3 和 TP Channel4 使用相同的源物理端口 2。

表 6-9 TPG 示例

TPG	TP Channel1 ( 目的, 源物理端口 1, UDP source port1, ... )
	TP Channel2 ( 目的, 源物理端口 1, UDP source port2, ... )
	TP Channel3 ( 目的, 源物理端口 2, UDP source port3, ... )
	TP Channel4 ( 目的, 源物理端口 2, UDP source port4, ... )

TPG 调度其管理的 TP Channel 发出 TP Packet，调度方式用户可自定义（如根据 TP Channel 队列深度等），调度到 TP Channel1 时，TP Packet 的 source port 域段将被设置为 source port1 并发出，依此类推；TPG 通过调度多个 TP Channel 充分均匀使用了 UBPU 多个物理端口的带宽。UB Switch 选路时，从一个 TP Channel 发出的流量因为具有包括 source port 域段在内相同的用于选路的元组，会经过相同的路径；因为从四个 TP Channel 发出的 TP Packet 会分别携带不同 source port 域段，可能会经过不同的路径，从而充分利用了网络的带宽。

### 6.5.1.3 RTP 控制逐包负载均衡

从 TP Channel 发出的 packet，在网络中会执行选路。在事务层指定事务操作可支持逐包负载均衡时，UB 传输层可以为每个 packet 分别设置不同的负载均衡因子值（见 5.2 节），控制一个 TP Channel 内发出的不同 TP Packet 选择不同的网络路径。

具体地，TP Sender 可为 TP Packet 设置不同的 UDP source port（IP/UDP 包格式下）或 LBF 字段（CNA 包格式下），以主动控制实现逐包负载均衡。

### 6.5.1.4 逐包负载均衡适配

#### 6.5.1.4.1 乱序接收适配逐包负载均衡

逐包负载均衡可能导致 TP Packet 乱序。UB 支持乱序接收，乱序接收程度即 PSN 乱序区间的大小，可基于 TP Channel 为粒度进行配置。乱序区间大小可在[128,256,512,1024,2048]中选择，单位为 TP Packet 个数。UB 支持通过设置 TP Sender 的最大发送窗口的方式限制单个 TP Channel 的在途 TP Packet 数目，将 TP Channel 的乱序度控制在可接受范围内，必要时可配合 UB 多 TP Channel 能力保证业务带宽。

#### 6.5.1.4.2 重传机制适配逐包负载均衡

重传机制 6.4.2 节中介绍了 UB 协议中支持设置快速重传和超时重传。在启用逐包负载均衡场景下，为了避免乱序 TP Packet 触发不必要的重传，TP Sender 可设置超时重传的方式进行重传；为了减少应答包的发送，用户可配置 TP Receiver 设置累计接收的乱序包数量作为返回 TPSACK 的触发条件。

## 6.5.2 CTP 多路径负载均衡

### 6.5.2.1 基于 Entity 负载均衡

CTP 提供灵活、轻量级的负载均衡机制。CTP 支持基于 Entity 的负载均衡，通过 Entity 灵活绑定多个端口，在多个端口间做负载均衡。在逐流负载均衡下，CTP Sender 可以通过为每个目的 Entity 指定 CNA 包头中的负载均衡因子 LBF 字段，控制 UBPU 网络层在 Entity 绑定的多个端口间的选择；在逐包

负载均衡下，CTP Sender 可以通过为每个 CNA 包头指定不同的 LBF 字段，或者根据 RoundRobin、端口负载情况选择合适的端口。

### 6.5.2.2 CTP 控制逐包负载均衡

同时，和 RTP 类似，逐包负载均衡机制可以通过 (i) CTP 指定负载均衡因子 LBF 字段，从而控制 packet 在 UB Switch 上的路径选择的方式实现，也可以通过 (ii) 在 UB Switch 启用逐包负载均衡的方式实现（见 5.3.2 节，本章节不做赘述）。

## 6.6 拥塞控制机制

### 6.6.1 RTP 拥塞控制机制

UB 传输层提供拥塞控制机制用于控制 TP Sender 的发送窗口或发送速率以避免或缓解网络拥塞。UB 支持多种拥塞控制算法，也支持用户自定义拥塞控制算法。拥塞控制算法在建立 TP Channel 时协商确定，TP Channel 的建立流程具体实现方式不在本规范定义。传输层端侧有 TPACK-CC、TPNAK-CC、TPSACK-CC 和 CNP 四种可以向发送侧反馈拥塞信号的 packet，其类型由 TPOpcode 指定，具体拥塞控制信息由 packet 中的 CETPH ( Congestion Extended Transport Header ) 包头来携带，不同拥塞控制算法拥有不同结构的 CETPH，CETPH 包头应符合 6.2.2 节相关要求。

传输层 TP Sender 可以基于发送窗口或者发送速率对发出的数据量进行控制。UB 传输层可选支持 LDPC ( Low Delay Control Protocol )，LDPC 为基于窗口的拥塞控制算法；UB 也可选支持 CNP 能力，可用以实现 DCQCN；UB 可选支持 UB 交换机主动队列管理拥塞控制算法 ( Confined Active Queue Management, CAQM )。除 UB 支持的几种拥塞控制算法外，UB 也支持用户自定义拥塞控制算法。

#### 6.6.1.1 基于窗口的拥塞控制

拥塞控制以调节发送窗口的方式控制网络拥塞时，该窗口被称为拥塞窗口(congestion window,cw)。拥塞窗口的单位在建立 TP Channel 时协商，默认以 1 字节为单位，其大小受到网络拥塞情况的影响。

##### 6.6.1.1.1 发送侧状态

TP Sender 需维护 CC ( Congestion Control ) Context，用于控制拥塞窗口，CC Context 的维护粒度为 TP Channel，即每条 TP Channel 都有单独的 CC Context。CC Context 中与拥塞控制相关的变量见表 6-10。其中数据量将传输层包头、传输层 Payload 与 ICRC 统计在内，与拥塞窗口单位一致。

表 6-10 CC Context 中与拥塞控制相关的变量

变量	全名	说明
cw	congestion window	用于控制 TP Sender 飞行 TP Packet 的窗口
inflight	inflight data size	已被 TP Sender 发送并假定仍在网络中的数据量
data_size_sent	data size sent	TP Sender 已经注入网络的数据量
data_size_rcvd	data size received	已被 TP Receiver 接收的数据量
aval_win	available window	TP Sender 剩余被允许发送的数据量

### 6.6.1.1.2 发送侧处理逻辑

TP Sender 发送数据前需要检查可用窗口(available window,aval\_win)是否足够, aval\_win 为 TP Sender 当前可以向网络中注入的数据量, 当 aval\_win 大于等于 TP Packet 所对应的数据量时数据才能发出。aval\_win 由拥塞窗口 ( cw ) 减去飞行流量 ( inflight ) 得到, 即  $aval\_win = cw - inflight$ 。

TP Sender 通过感知网络拥塞状态调整 cw 值。一般地, TP Sender 在检测到拥塞时执行减小 cw 操作, 在检测到拥塞解除时执行增加 cw 操作。TP Sender 通过 TPACK-CC/TPNAK-CC/TPSACK-CC 中 CETPH 携带的拥塞信息检测拥塞状态。例如当 TP 启用 LDCP 时, TP Sender 通过 CETPH.Ce\_Seq 获取被打标的 TP Packet 数量, 从而减少 cw 值。

TP Sender 需要统计飞行流量 ( inflight ), 飞行流量指的是 TP Sender 已经发送到网络中、但是还未被确认的数据量。inflight 的更新分别发生在 TP Sender 发出 TP Packet 和收到 TPACK-CC/TPNAK-CC/TPSACK-CC 时。当发送出新的 TP Packet 时, inflight 变量增加 TP Packet 大小; 在收到 TPACK-CC/TPNAK-CC/TPSACK-CC 时, inflight 变量减少其确认接收的 TP Packet 大小; 特别地, 在触发重传时, inflight 变量需相应减少。

为了计算 inflight, TP Sender 需要维护 data\_size\_sent 变量, 初始化为 0。每发出一个 TP Packet, 该变量增加对应的数据量。TP Sender 和 TP Receiver 分别维护 data\_size\_rcvd, 用于记录已经被 TP Receiver 确认接收的数据量。TP Receiver 产生 TPACK-CC 时, data\_size\_rcvd 通过 CETPH 的 Ack\_seq 域段返回, 用于更新 TP Sender 维护的 data\_size\_rcvd。TP Sender 所维护的 inflight 由 data\_size\_sent 减去 data\_size\_rcvd 得到。

UB 支持选择性重传和 GoBackN 两种重传算法, 在不同重传算法下 inflight 变量的更新方式有所不同。

在使用 GoBackN 重传时: 当 TP Sender 发起 GoBackN 重传时, data\_size\_sent 回退至 TPNAK-CC 中的 CETPH.Ack\_seq, 此时 inflight 会减为 0, 即重传时放弃所有飞行流量。

在使用选择性重传时: 当 TP Sender 发起选择性重传时, data\_size\_sent 需要额外减去 TPSACK-CC 指示的需重传的 TP Packet 大小, 用于 inflight 的更新。

特别地, 当 TP Sender 检测到超时重传时, 直接重置 inflight 为 0。等到下一个 TPACK-CC/TPNAK-CC/TPSACK-CC 返回时, 恢复按照以上逻辑执行 inflight 计算。

### 6.6.1.1.3 交换侧状态及处理逻辑

UB Switch 对经历拥塞的 TP Packet 打上拥塞标记，具体机制应符合网络层 5.3.5 节相关要求。

### 6.6.1.1.4 接收侧状态

TP Receiver 维护 data\_size\_recv 变量，初始化为 0，每正确收到一个 TP Packet（通过 PSN 校验），该变量增加 TP Packet 大小。

### 6.6.1.1.5 接收侧处理逻辑

若 TP Receiver 正确接收到 TP Packet，未触发重传，则生成 TPACK-CC，将 data\_size\_recv 通过 TPACK-CC 中的 CETPH.Ack\_seq 域段返回 TP Sender。

UB 支持选择性重传和 GoBackN 两种重传算法，在不同重传算法下 TP Receiver 处理方式有所不同。

在使用 GoBackN 重传时：当 TP Receiver 收到乱序 TP Packet 时，data\_size\_recv 不增加，生成 TPNAK-CC，同时 data\_size\_recv 通过 TPNAK-CC 中的 CETPH.Ack\_seq 域段返回给 TP Sender。

在使用选择性重传时：若 TP Receiver 收到乱序 TP Packet，但 TP Packet 处于乱序区间内，则正常接收该 TP Packet，data\_size\_recv 增加对应 TP Packet 大小，生成 TPSACK-CC，同时 data\_size\_recv 通过 TPSACK-CC 的 CETPH.Ack\_seq 域段返回 TP Sender。

TPACK-CC/TPNAK-CC/TPSACK-CC 中的 CETPH 除携带基本的 Ack\_seq 域段外，在不同拥塞控制算法下，还会携带其它拥塞信息，相应的 CETPH 格式应符合 6.2.2 节相关要求。

### 6.6.1.2 基于速率的拥塞控制

与基于窗口的拥塞控制相比，基于速率的拥塞控制有以下特点：

1. TP Sender 根据应答包中带回的网络拥塞信息，以一定的速率发送 TP Packet。因为 TP Sender 不设置发送窗口，不能精确控制网络的飞行中 TP Packet。
2. 因为没有发送窗口的限制，不需要计算 inflight 和 aval\_win，因此应答包中不需要携带 Ack\_seq，TP Sender 的 CC Context 中不需要维护和序列号相关的变量。资源占用量相对较少。

### 6.6.1.3 基于交换机主动队列管理的拥塞控制

UB 可选支持基于交换机主动队列管理的拥塞控制（Confined Active Queue Management, CAQM），其采用端网结合（TPEP 和 UB Switch 配合）的方式，UB Switch 可以根据自身拥塞状况，决策是否批准 TP Sender 的窗口增长请求。在 TP Sender，随 TP Packet 携带“窗口增加的请求”，网络路径上的每个 UB Switch 视自身的空闲转发能力决策是否批准该请求，决策的结果随 TP Packet 到达 TP Receiver。TP Receiver 把决策结果通过 TPACK-CC/TPNAK-CC/TPSACK-CC 返回至 TP Sender，TP Sender 据此更新

窗口。CAQM 可以配合基于窗口或者速率的拥塞控制使用，本小节中描述 CAQM 与基于窗口的拥塞控制配合的逻辑。

### 6.6.1.3.1 CAQM 域段

在 TP Sender，随 TP Packet 携带“窗口增加的请求”需要三个域段，分别是拥塞标记位(1 bit)、请求增加指示位(1 bit)和 Hint 域段(8 bits)，其中拥塞标记位 ( C 位 ) 表示该 TP Packet 在传输过程中是否经历过拥塞，请求增加指示位 ( I 位 ) 用于指示 Hint 域段是否有效，当 I 位的值为 1 时，Hint 域段携带的是 TP Sender 请求的拥塞窗口增加量 ( 见 5.3.5.3 节 )。此三个域段在 TP Packet 中由网络层 CCI 域段来承载。UB Switch 依据这三个域段携带的信息运行 CAQM 算法，算法得出的决策结果由 C 位和 I 位表示。这三个域段通过 TPACK-CC/TPNAK-CC/TPSACK-CC 的 CETPH 返回给 TP Sender。

### 6.6.1.3.2 CAQM 发送侧流程

TP Sender 发送 TP Packet 时，随 TP Packet 携带窗口增加请求。具体地，C 位清零，I 位置 1，在 Hint 域填充其期望的窗口增加量 ( 如果不需要增加，I 位置 0 即可 )。

TP Sender 接收到返还的应答包 ( TPACK-CC/TPNAK-CC/TPSACK-CC ) 时，根据 CETPH 中的 C、I、Hint 信息调整 cw 窗口的大小，窗口调整规则如表 6-11 所示：

表 6-11 窗口调整规则

C	I	TP Sender 行为
大于 0	0	降低拥塞窗口
大于 0	1	窗口同时有增量和减量，增量由大小由 CETPH.Hint 表示，减量受 CETPH.C 影响
0	1	增加拥塞窗口，大小由 CETPH.Hint 表示
0	0	保持窗口不变

### 6.6.1.3.3 CAQM 交换侧流程

TP Packet 经过 UB Switch 时，UB Switch 根据自身拥塞情况，更新网络层 CCI 中的 C、I 控制比特，具体机制应符合 5.3.5 节相关要求。

### 6.6.1.3.4 CAQM 接收侧流程

TP Receiver 处理方式分两种情况：在 TPACK 未聚合场景下，TP Receiver 直接将收到的 TP Packet 的网络层包头 NTH.CCI 域段中的 C、I、Hint 域段通过应答中 ( TPACK-CC/TPNAK-CC/TPSACK-CC ) 的 CETPH 头部返回 TP Sender。在 TPACK 聚合场景下，TP Receiver 需在本地维护 Hint、C、I 变量用于聚合拥塞信息。TP Receiver 每收到一个 TP Packet，若 NTH.CCI 域段的 C 域段为 0 且 I 域段为 1 时，则本

地 Hint 累加，且将本地 I 变量设置为 1；若 NTH.CCI 的 C 域段为 1，则本地 C 变量累加 1。在 TP Receiver 返回应答时将本地维护的 C、I、Hint 变量通过应答包对应域段带回 TP Sender，并将 TP Receiver 本地维护的 C、I、Hint 变量清零。

#### 6.6.1.4 拥塞控制适配多路径负载均衡

对于 RTP 而言，拥塞控制上下文一般以 TP Channel 为粒度进行维护，而在逐包负载均衡下，共享同一个 TP Channel 的 TP Packet 可能经历不同的网络路径。在逐包负载均衡下，如果拥塞控制的速率调整对网络拥塞过于敏感而产生过度响应，单条网络路径的拥塞可能会引起 TP Channel 内所有流量的发送速率降低，从而可能导致网络欠吞吐。在逐包负载均衡下，建议将拥塞控制和负载均衡协同设计，例如拥塞控制仍以 TP Channel 为粒度进行拥塞控制上下文的维护，但拥塞控制不对 TP Channel 内 TP Packet 经历的每条网络路径的拥塞进行精细化控制，而将网络当做一个整体进行控制，识别流量在所有可选路径上的整体拥塞状态，在所有可选路径发生拥塞时进行降速。

启用逐流负载均衡的场景下，选择相同 TP Channel 的 TP Packet 将经过相同的网络路径。因此，TPG 中的每条 TP Channel 可单独维护上下文并进行拥塞控制。

### 6.6.2 CTP 拥塞控制机制

CTP 开启拥塞控制时，以{目的 Entity, VL}（其中 VL 为物理链路上独立的逻辑通道）为粒度进行粗粒度的拥塞控制。轻量级模式下无传输层应答，支持以单独拥塞反馈包（如 CNP）带回拥塞信号。

CTP 适配逐包负载均衡使用时和 RTP 适配逐包负载均衡类似，可将网络当做一个整体进行拥塞控制。

## 6.7 传输流程

### 6.7.1 RTP 传输流程

#### 6.7.1.1 TP Sender 传输流程

TP Sender 发送 TP Packet 的处理流程如下：

1. TP Channel 选择

传输层从事务层接收事务操作，通过事务信息得到 TPN 或 TPGN（TPG 编号），选中相应的 TP Channel 或 TPG 来传输数据；若使用 TPG，还需根据 TPG Context，以一定的策略（见 6.5.1.1 节）根据事务操作的粒度选择一条合适的 TP Channel 传输数据。

## 2. PSN 生成

TP Sender 会将事务层下发的操作以传输层 MTU ( Maximum Transmission Unit ) 的大小切分为 TP Packet 进行传输。TP Sender 必须在 TP Packet 的 RTPH 中设置 PSN。

TP Sender 根据当前维护的 PSN 值为 TP Packet 进行编号。TP Sender 每生成一个非重传的 TP Packet 后，当前 PSN 的值加一。

根据以上规则，设当前 PSN 为 curr\_PSN，下一个 PSN 为 next\_PSN，则  $next\_PSN = (curr\_PSN + 1) \bmod 16M$ 。再次生成 TP Packet 包时，令  $curr\_PSN = next\_PSN$ ，然后继续计算出 next\_PSN。

## 3. 操作码生成

RTPH 中操作码 TPOopcode 长度为 8bits，其中 TPOopcode[6:0] 表示 packet 类型，该场景下类型为 TP Packet，因此 TPOopcode[6:0] 的值应设置为 1；TPOopcode[7]（记为 last 标记）表示该 TP Packet 是否为当前事务操作的最后一个 TP Packet，last 标记为 0 时，此 TP Packet 非当前操作的最后一个 TP Packet；last 标记为 1 时，此 TP Packet 为当前事务操作的最后一个 TP Packet。

## 4. 载荷生成

如果操作码指定了当前 TP Packet 为非当前操作的最后一个 TP Packet，则传输层有效载荷（指事务层数据，不包含事务层包头）的长度必须等于一个传输层 MTU，如果操作码指定了当前 TP Packet 为当前操作的最后一个 TP Packet，则有效载荷的长度必须介于 0 至传输层 MTU 之间。

当事务操作对应的有效载荷总长度超过一个 MTU，即传输层需要执行切包时，最后一个 TP Packet 的有效载荷长度必须大于 0。

### 6.7.1.1.2 TP Sender 接收处理传输层应答

传输层应答类型包含 TPACK/TPNAK/TPSACK 三种类型，TP Sender 处理应答包逻辑见 6.4.2 节。

### 6.7.1.2 TP Receiver 传输流程

#### 6.7.1.2.1 TP Receiver 接收 TP Packet

TP Receiver 接收 TP Packet 的处理流程如下：

##### 1. TP Packet 接收

根据 TP Packet 中的 RTPH. DstTPN 找到对应的 TP 上下文，接收对应 TP Packet。

##### 2. PSN 校验

TP Receiver 根据维护的 EPSN 字段与 TP Packet 携带的 PSN 进行校验，具体处理逻辑见 6.4.2 节。

##### 3. 提交事务层

将 TP Packet 中的 Payload 提交事务层。

#### 4. 操作接收完成检查

TP Receiver 需检查收到的 TP Packet 是否为当前操作的最后一个 TP Packet, 该信息通过 RTPH 中的 TPOpcode[7] ( 即 last 标记 ) 来确定, 当前操作的编号由 RTPH 中的 TPMSN 确定。若 last 标记为 0, 说明 TP Packet 不是当前操作的最后一个 TP Packet。若 last 标记为 1, 则说明 TP Packet 是当前操作的最后一个 TP Packet。若已接收到当前操作的最后一个 TP Packet, 且之前的所有 TP Packet 也都接收完毕, 说明当前操作已被完整接收, 传输层通知事务层编号为 TPMSN 的操作接收完成, 事务层可以执行该操作。事务层执行操作成功后, 若返回 TAACK 给 TP Receiver, 传输层将该 TAACK 作为 TP Packet 发出; 注意此时该 TP Receiver 转换为 TP Sender 的角色发送载荷为 TAACK 的 TP Packet, 对端传输层接收后会执行其 TP Receiver 逻辑接收 TP Packet 并发出 TPACK/TPNAK/TPSACK。

#### 6.7.1.2.2 TP Receiver 发送传输层应答

三种传输层应答的生成逻辑见 6.4.2 节。特别地, 传输层应答还可用于携带事务层的应答信息 ( 见 7.3.1 节 )。

### 6.7.2 CTP 传输流程

#### 6.7.2.1 TP Sender 发送 TP Packet

TP Sender 发送 TP Packet 的处理流程如下:

##### 1. 操作码生成

CTPH 中操作码 TPOpcode 长度为 2bits, CTP 下, 事务层下发的事务操作均可用一个 TP Packet 承载, 设置 TPOpcode[1:0]的值为 0x0。

##### 2. 载荷生成

有效载荷的长度必须介于 0~传输层 MTU 之间。

#### 6.7.2.2 TP Receiver 接收 TP Packet

TP Receiver 接收 TP Packet 的处理流程如下:

##### 1. TP Packet 接收

TP Receiver 接收对应的 TP Packet。

##### 2. 提交事务层

CTP 模式下, 每个 TP Packet 都携带一个完整的事务操作。因此 TP Receiver 接收到 TP Packet 时, 直接通知事务层执行该 TP Packet 对应的事务操作。

##### 3. 操作接收完成检查 ( 事务层 )

事务层执行事务操作成功后,若返回 TAACK 给 TP Receiver, 传输层将该 TAACK 作为 TP Packet 发出; 注意此时该 TP Receiver 转换为 TP Sender 的角色发送载荷为 TAACK 的 TP Packet, 对端传输层接收后会执行其 TP Receiver 逻辑接收 TP Packet。

## 6.8 传输层事务层交互流程

本章节描述事务层和传输层的交互逻辑。

在事务层,发起事务操作的一端称为 Initiator,接收事务操作的一端称为 Target。在传输层,发送 TP Packet 的一端称为 TP Sender,接收 TP Packet 的一端称为 TP Receiver。TP Sender 与 TP Receiver 的位置在事务流程中可能会发生改变:考虑 UBU A 作为 Initiator 和 UBU B 作为 Target 的场景,UBU A 的传输层作为 TP Sender 发送承载事务操作的 TP Packet,UBU B 的传输层作为 TP Receiver 接收该 TP Packet;然而,UBU B 传输层需要作为 TP Sender 回复承载 TAACK 的 TP Packet,此时 UBU A 传输层将作为 TP Receiver。因此为了避免混淆,在本章节统一使用 Initiator 传输层、Target 传输层来表示传输层。

### 6.8.1 事务层 ROI、ROT 模式下交互流程

本章节以 Write 事务为例,描述事务层 ROI、ROT 模式下 RTP 传输层传输流程如图 6-34 所示(注:此处的流程示例中事务操作大小为 2 个 TP Packet,且 Initiator 传输层和 Target 传输层发送前协商的起始 PSN 为 m 和 n)。事务层的应答包为 TAACK/TANAK,值得说明的是,从传输层看,事务层下发的事务操作以及 TAACK 都由若干个 TP Packet 组成,因而在 Initiator 传输层收到承载事务层 TAACK 的事务操作时,应回复 TPACK。

1. Initiator 传输层接收事务层下发的操作并根据传输层 MTU 进行切包和封装。假设此事务操作可以用两个 TP Packet 承载,TP Sender 发送 PSN=m 和 PSN=m+1 的两个 TP Packet。
2. Target 传输层根据收到的 TP Packet 的 PSN 判断 PSN=m 的 TP Packet 为正序包,因此回复携带 PSN=m 的 TPACK,之后又收到 PSN=m+1 的正序包,同理回复携带 PSN=m+1 的 TPACK。
3. 因为本示例中事务操作用两个 TP Packet 承载,所以接收到 PSN=m+1 的 TP Packet 意味着事务操作的成功接收。Target 传输层上报 Target 事务操作成功接收,Target 事务层回复 TAACK。Target 传输层发送承载 TAACK 的 PSN=n 的 TP Packet。
4. Initiator 传输层收到 PSN=m+1 的 TPACK,即指示 PSN 为 m+1 及之前的 TP Packet 已被成功接收,更新本地发送状态。
5. Initiator 传输层收到承载 TAACK 的 PSN=n 的 TP Packet 后,回复 PSN=n 的 TPACK。因为这个 TP Packet 接收成功后,对应的事务操作(TAACK)也接收成功,Initiator 传输层上报 Initiator 事务层。

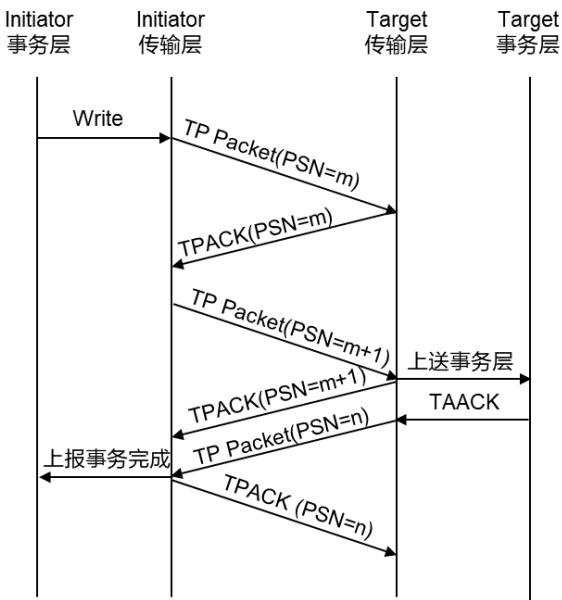


图 6-34 Write 操作流程图 ( ROI 和 ROT 模式 )

因为 CTP 传输层不生成 TPACK，在事务层 ROI 和 ROT 模式下由事务层生成 TAACK。

### 6.8.2 事务层 ROL 模式下交互流程

事务层 ROL 模式与其它事务层模式不同，事务层 ROL 模式不生成独立的事务层应答包，而由传输层应答包捎带事务层应答。此时传输层应答包不仅承载了对 TP Packet 的应答，也承载了事务操作的应答信息，所以传输层应答需要等待事务层的响应结果。本章节以 Write 事务为例，描述事务层 ROL 模式下 RTP 传输层传输流程，如图 6-35 所示。

1. Initiator 传输层接收事务层下发的事务操作并进行切包和封装。此事务操作用两个 TP Packet 承载，TP Sender 发送携带 PSN=m 和 PSN=m+1 的 TP Packet ( Initiator 传输层的起始 PSN 为 m )。
2. Target 传输层根据收到的 TP Packet 的 PSN 判断 PSN=m 的 TP Packet 为正序包，因此回复携带 PSN=m 的 TPACK。当收到 PSN 为 m+1 的 TP Packet 时，与 ROI 等其它事务层模式不同的是，此时 TP Receiver 不会立即回复 TPACK。
3. 因为 PSN=m+1 为承载事务操作的最后一个 TP Packet，此 TP Packet 的成功接收也意味着事务操作的成功接收。Target 传输层上报 Target 事务层事务操作成功接收，Target 事务层处理完成通知 Target 传输层。Target 传输层生成 PSN=m+1 的 TPACK，此应答同时承载了 Target 传输层和事务层对 Initiator 传输层和事务层的应答。
4. Initiator 传输层接收到 PSN=m+1 的 TPACK，即指示 PSN=m 和 PSN=m+1 的两个 TP Packet 已被成功接收，因此更新本地发送状态。因为此 TPACK 同时承载了对 Initiator 事务层的应答，所以 Initiator 传输层会通知事务层事务完成情况。

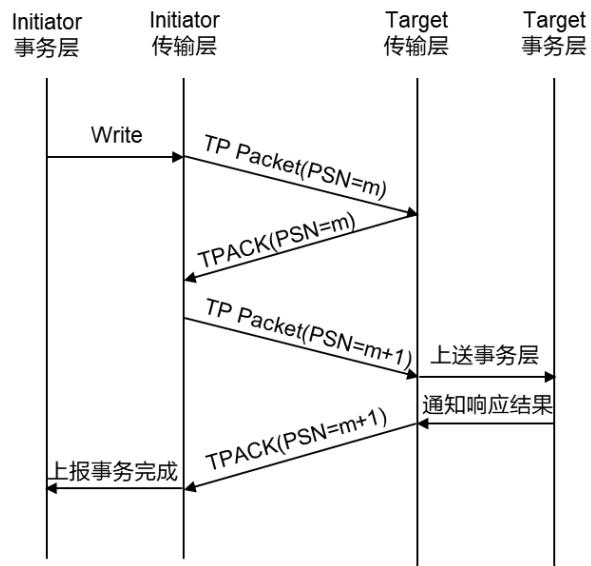


图 6-35 Write 操作流程图 (ROL 模式)

因为 CTP 传输层不生成 TPACK，在事务层 ROL 模式下由事务层生成 TAACK。

# 7 事务层

## 7.1 概述

事务层位于传输层之上，向上层提供多种事务服务以及事务操作能力。事务层接收上层编程模型（包括 Load/Store 同步访问、URMA 异步访问）等的事务调用，联合下层协议完成 Initiator、Target 之间的事务操作。为优化传输效率，针对 Load/Store 同步访问发起的事务操作，如内存读写，事务层可选使用 TP Bypass 模式以及精简包格式。

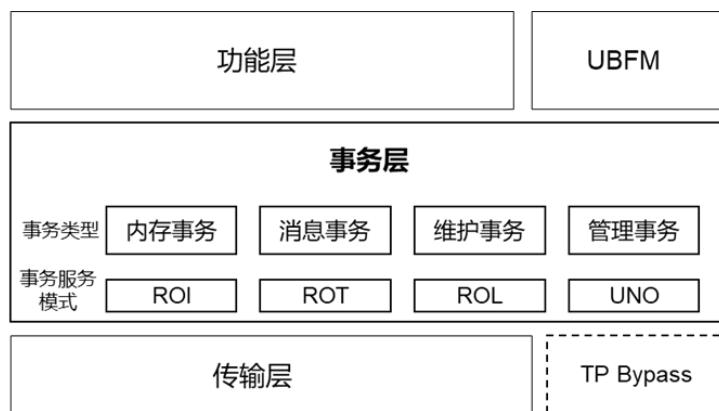


图 7-1 事务层概览

事务层支持内存事务、消息事务、维护事务以及管理事务四种事务类型，每种事务类型分为多种事务子类型，通过不同的事务操作码，即 TAOpcde 区分，见 7.4 节。一次完整的事务操作包含至少一次事务请求以及可能的事务响应。

事务层提供可靠事务服务以及不可靠事务服务两种模式。针对可靠事务服务，事务层除要求下层协议保证事务层包可靠传输外，针对可处理的异常（如资源不足），事务层支持事务重传，针对不能处理的异常，事务层应将事务状态上报给上层，由上层完成异常处理；针对不可靠事务服务，事务层不要求下层协议提供可靠性，且事务层不会对异常事务做重传处理。在多个事务需要处理时，有些事务和其它事务之间有特定的顺序要求，包括执行顺序以及完成顺序。事务层通过多种机制保证有顺序要求的事务按照指定顺序执行或者完成。

根据是否提供可靠性以及执行序保证，事务层提供 Reliable and Ordered by Initiator ( ROI )、Reliable and Ordered by Target( ROT )、Reliable and Ordered by Lower layer( ROL )、Unreliable and Non-Ordering ( UNO ) 4 种事务服务模式，供上层调用。每种事务操作可使用的事务服务模式见 7.4 节。不同的事务服务模式由于在可靠以及保序方面的要求不同，对下层协议要求也不同，配合关系见 7.3.4 节。

为保障事务操作的安全性，事务层交互过程中支持携带访问凭据，用于访问权限控制。校验机制包括对内存操作的权限校验、对消息对象访问的权限校验等。

UB 事务层具有以下核心特点：

1. 统一的事务操作，支持被不同编程模型使用。
2. 提供多种事务服务模式，满足事务操作在可靠、保序上的差异化要求。
3. 提供安全机制，保障事务操作安全执行。
4. 针对 Load/Store 同步访问发起的事务，可采用 TP Bypass 模式以及精简包格式，提升传输效率。

## 7.2 事务层包头 (TAH)

### 7.2.1 Basic Transaction Header ( BTAH )

BTAH 是所有事务请求必须包含的事务层包头。BTAH 包括完整格式以及精简格式。

完整 BTAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
TAOpcode								TAvver	EE_bits		TV_EN	Poison	RSVD		UD_Flag	INI_TASSN															
No_TAACK	ODR	MT_EN	FCE	Retry	Alloc	RSVD	E_bit	INI_RC_Type	INI_RC_ID																						

图 7-2 完整 BTAH 格式

精简 BTAH 格式如下：

Byte0								Byte1								Byte2								Byte3								
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	
TAOpcode								TAvver	EE_bits		TV_EN	Poison	RSVD		UD_Flag	INI_TASSN																

图 7-3 精简 BTAH 格式

BTAH 包含域段如下所示：

表 7-1 BTAH 域段

域段名	位宽( bits )	完整 BTAH 域段描述	精简 BTAH 域段描述
TAOpcode	8	Transaction Opcode，表示当前事务的操作类型，具体编码如下：	同左

域段名	位宽( bits )	完整 BTAH 域段描述	精简 BTAH 域段描述
		<ul style="list-style-type: none"> <li>• 0x0: Send</li> <li>• 0x1: Send_with_immediate</li> <li>• 0x2: Reserved</li> <li>• 0x3: Write</li> <li>• 0x4: Write_with_immediate</li> <li>• 0x5: Write_with_notify</li> <li>• 0x6: Read</li> <li>• 0x7: Atomic_compare_swap</li> <li>• 0x8: Atomic_swap</li> <li>• 0x9: Atomic_store</li> <li>• 0xA: Atomic_load</li> <li>• 0xB: Atomic_fetch_add</li> <li>• 0xC: Atomic_fetch_sub</li> <li>• 0xD: Atomic_fetch_and</li> <li>• 0xE: Atomic_fetch_or</li> <li>• 0xF: Atomic_fetch_xor</li> <li>• 0x10: Management</li> <li>• 0x11-0x13: 见 ATAH</li> <li>• 0x14: Write_with_be</li> <li>• 0x15: Prefetch_tgt</li> <li>• 0x16: Reserved</li> <li>• 0x17: Writeback</li> <li>• 0x18: Writeback_with_be</li> <li>• Others: Reserved</li> </ul>	
TAver	2	Transaction version, Transaction 协议版本号, 当前协议应设置为 0。	同左
EE_bits	2	<p>Execution Environment bits, 执行环境标识位, 用于标识事务来源的执行环境的安全属性:</p> <ul style="list-style-type: none"> <li>• 2'b00: Reserved。</li> <li>• 2'b01: Non-TEE。</li> <li>• 2'b10: Reserved。</li> <li>• 2'b11: TEE。</li> </ul>	同左
TV_EN	1	<p>TVETAH enable, 表示 TVETAH 头的启用情况:</p> <ul style="list-style-type: none"> <li>• 1'b0: 不携带 TVETAH 头。</li> <li>• 1'b1: 携带 TVETAH 头。</li> </ul>	同左

域段名	位宽( bits )	完整 BTAH 域段描述	精简 BTAH 域段描述
Poison	1	表示 Payload 是否存在 poison: <ul style="list-style-type: none"><li>• 1'b0: 不存在。</li><li>• 1'b1: 存在。</li></ul>	同左
UD_Flag	1	User Defined Flag, 指示是否携带 UDETAH 头: <ul style="list-style-type: none"><li>• 1'b0: 不携带 UDETAH 头。</li><li>• 1'b1: 携带 UDETAH 头。</li></ul>	同左
INI_TASSN	16	Initiator Transaction Segment Sequence Number ( TASSN ), 表示事务请求编号。 在 Target 生成事务响应时, 把此域段带回给 Initiator, 以便 Initiator 识别哪个事务得到应答。如果事务被切片, 每个切片分配一个 TASSN。  CTP 场景下, 每个切片包含一个包, 占据一个 TASSN。	同左
No_TAACK	1	是否返回 Transaction Acknowledgement ( TAACK ), 仅传输层使用 CTP 时有效: <ul style="list-style-type: none"><li>• 1'b0: 需要返回 TAACK。</li><li>• 1'b1: 不需要返回 TAACK。</li></ul>	无此域段
ODR	3	事务请求的 Order 属性标识。 ODR[1:0]表示事务执行序属性: <ul style="list-style-type: none"><li>• 2'b00: No Order ( NO ), 表示当前事务和其它事务无保序要求。</li><li>• 2'b01: Relax Order ( RO ), 表示当前事务与后续带 SO 标记的事务有保序要求, 带 SO 标记的事务不能先于前面带 RO 标记的事务执行。</li><li>• 2'b10: Strong Order ( SO ), 表示当前事务与前面带 RO 或 SO 标记的事务有保序要求, 不能先于前面带 RO 或 SO 标记的事务执行。</li><li>• 2'b11: Reserved。</li></ul> ODR[2]表示事务完成序属性: <ul style="list-style-type: none"><li>• 1'b0: 无完成序要求。</li><li>• 1'b1: 有完成序要求。</li></ul>	无此域段

域段名	位宽( bits )	完整 BTAH 域段描述	精简 BTAH 域段描述
MT_EN	1	MTETAH enable, 表示 MTETAH 头的启用情况： ● 1'b0：不携带 MTETAH 头。 ● 1'b1：携带 MTETAH 头。	无此域段
FCE	1	Fast Completion Event, 用于 Target 判断 Complete Queue Element ( CQE ) 是否产生完成事件： ● 1'b0：不产生完成事件。 ● 1'b1：产生完成事件。	无此域段
Retry	1	指示请求为首次发送还是重传发送： ● 1'b0：首次发送。 ● 1'b1：重传发送。	无此域段
Alloc	1	申请 Target Sequence Context ID ( SCID ) , 仅用于 ROT 动态模式的首包中： ● 1'b0：不申请 SCID。 ● 1'b1：申请 SCID。	无此域段
E_bit	1	Exclusive bit, 用于排他性校验： ● 1'b0：非 Exclusive 模式。 ● 1'b1：Exclusive 模式。	无此域段
INI_RC_Type	2	Initiator Requester Context Type： ● 2'b00 和 2'b01: Initiator 使用 Send Queue ( SQ ) 。 ● 2'b10 : Initiator 使用 Target Sequence Context ( SC ) 资源。 ● 2'b11: Reserved。	无此域段
INI_RC_ID	20	Initiator Requester Context ID： ● 若 INI_RC_Type 为 2'b00 或 2'b01，此域段表示 Requester Context ID( RCID ) , Target 生成事务响应时，把此域段带回给 Initiator，以便 Initiator 用来索引正确的 SQ。 ● 若 INI_RC_Type 为 2'b10，此域段表示 SCID，Target 产生事务响应时，从 SC 中取出 RCID, 填到 ATAH 中，以便 Initiator 用来索引正确的 SQ。	无此域段

域段名	位宽( bits )	完整 BTAH 域段描述	精简 BTAH 域段描述
		<ul style="list-style-type: none"> <li>若事务为资源管理消息，此域段表示 message queue index ( 上限根据实现决定 )，Target 产生事务响应时，把此域段带回给 Initiator，以便 Initiator 用来索引正确的 message queue。</li> </ul>	

不同事务操作使用的包格式如下表所示：

表 7-2 不同事务操作包格式

事务类型	TAOpcode	事务子类型	事务层包格式
内存事务	Write	0x3	BTAH,[UDETAH],MAETAH,[TVETAH],Payload
		0x5	BTAH,[UDETAH],MAETAH,NTFETAH,[TVETAH], Payload
		0x14	BTAH,[UDETAH],MAETAH,[TVETAH],BEETAH, Payload
		0x17	BTAH,[UDETAH],MAETAH,[TVETAH],Payload
		0x18	BTAH,[UDETAH],MAETAH,[TVETAH],BEETAH, Payload
	Read	0x6	BTAH,[UDETAH],[TAIDETAH],[OFSTETAH],MAETAH,[TVETAH]
		0x7	BTAH,[UDETAH],[TAIDETAH],MAETAH,[TVETAH],Payload
		0x8	
		0x9	
		0xA	
		0xB	
		0xC	
		0xD	
		0xE	
		0xF	
消息事务	0x0	Send	BT AH,[UDETAH],MTETAH,OFSTE TAH,[TVETAH],Payload
	0x1	Send_with_immediate	BT AH,[UDETAH],MTETAH,OFSTE TAH,IMMAETAH,[TVETAH], Payload
	0x4	Write_with_immediate	BT AH,[UDETAH],MTETAH, MAETAH,IMMETAH,[TVETAH], Payload

事务类型	TAOpcode	事务子类型	事务层包格式
维护事务	0x15	Prefetch_tgt	BTAH, [UDETAH],MAETAH
管理事务	0x10	Management	BTAH,[UDETAH],MGMTETAH, Payload
Else	RSVD		

注：[\*]为可选包头，是否携带见 7.4 节。

### 7.2.2 AcknowledgeTransaction Header (ATAH)

ATAH 是所有事务响应必须包含的事务层包头。包括完整格式以及精简格式，和 BTAH 格式对应，即如果 BTAH 采用完整格式，ATAH 也应采用完整格式，反之亦然。

完整 ATAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
TAOpcode								TAver	RSVD				SV	Poison	RSVD	INI_TASSN															
RSPST	RSPINFO			RSVD	INL_RC_Type		INI_RC_ID																								

图 7-4 完整 ATAH 格式

精简 ATAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
TAOpcode								TAver	Status	RSVD	Poison	RSVD	INI_TASSN																		

图 7-5 精简 ATAH 格式

各域段描述如下：

表 7-3 ATAH 域段

域段名	位宽( bits )	完整 ATAH 域段描述	精简 ATAH 域段描述
TAOpcode	8	表示当前事务的响应类型，具体编码如下： • 0x11: Transaction ACK ( TAACK ) • 0x12: Read_response	同左

域段名	位宽( bits )	完整 ATAH 域段描述	精简 ATAH 域段描述
		<ul style="list-style-type: none"> <li>• 0x13: Atomic_response</li> </ul>	
TAver	2	Transaction 协议版本号, 当前协议应设置为 0。	同左
Status	2	Reserved	<p>用于表示远端鉴权结果, 编码如下:</p> <ul style="list-style-type: none"> <li>• 2'b00 : 表示远端鉴权 Pass, 数据读写正确完成。</li> <li>• 2'b01 : 表示远端鉴权 Fail。</li> <li>• Others: Reserved。</li> </ul>
SV	1	<p>SCID Valid。 ROT 动态模式下是否申请到 Target SCID。如果此 bit 为 1, 申请到 ROT 资源, 则 SCID 有效, 携带在INI_TASSN 中。</p>	Reserved
Poison	1	<p>带数据的包该 bit 有效, 表示 Payload 是否存在 poison:</p> <ul style="list-style-type: none"> <li>• 1'b0: 不存在。</li> <li>• 1'b1: 存在。</li> </ul>	同左
INI_TASSN	16	<p>Initiator Transaction Segment Sequence Number:</p> <ul style="list-style-type: none"> <li>• SV 为 0 时, 表示 Initiator TASSN, 应对应事务请求的 TASSN。</li> <li>• SV 为 1 时, 该域段代表 ROT 模式时申请到的 Target SCID。</li> </ul>	Initiator Transaction Segment Sequence Number, 表示 Initiator TASSN, 应对应事务请求的 TASSN。
RSPST	3	<p>Response Status, 事务响应状态, 和 RSPINFO 搭配使用, 编码如下:</p> <ul style="list-style-type: none"> <li>• 3'b000: Successful Completion (SC)</li> <li>• 3'b001: Receiver Not Ready ( RNR ), Target 接收资源不足, 此时 RSPINFO 表示 RNR_Timer。</li> <li>• 3'b010: Page Fault, Target 出现 Page Fault, 此时 RSPINFO 表示 RNR_Timer。当出现 Page Fault 时, 有 2 种处理策略。策略 1: 不等 Page Fault 处理完, 回异常 TAACK, 告知发送端重传, 此策略下 RNR_Timer 通常大于 0。</li> </ul>	无此域段

域段名	位宽( bits )	完整 ATAH 域段描述	精简 ATAH 域段描述
		<p>策略 2: Page Fault 处理完, 回异常 TAACK, 告知发送端重传, 此策略下 RNR_Timer 为 0。</p> <ul style="list-style-type: none"> <li>• 3'b011: Completer Error。</li> <li>• 3'b101: 包正确接收, 但未写入介质。</li> <li>• Others: Reserved。</li> </ul> <p>注: 3'b000 和 3'b101 的差异是 3'b000 表示 Payload 已经写入内存, 而 3'b101 表示 Payload 已经交给介质控制器, 如内存控制器, 但并未保证已写入介质。</p>	
RSPINFO	5	<p>Response Information, 事务响应信息, 具体编码如下:</p> <ul style="list-style-type: none"> <li>• RSPST=3'b000 或 3'b101: 该域段表示 TAACK 聚合个数, 为 0 时表示此 TAACK 无聚合其它 TAACK, 非 0 时表示聚合其它 TAACK 的个数。</li> <li>• RSPST=3'b001 和 3'b010: 该域段表示 RNR_Timer 编码。RNR_Timeout = 2us * 2^RNR_Timer, RNR_Timer 有效编码范围为 [0,19], 其它编码 Reserved。</li> <li>• RSPST=3'b011: 该域段表示返回错误应答, 错误类型为: <ul style="list-style-type: none"> <li>- 5'b00001 表示 Unsupported Request。</li> <li>- 5'b00010 表示 Remote Abort。</li> </ul> </li> <li>• Others: Reserved。</li> </ul>	无此域段
INI_RC_Type	2	<p>Initiator Requester Context Type:</p> <ul style="list-style-type: none"> <li>• 2'b00 和 2'b01: Initiator 使用 Send Queue ( SQ )。</li> <li>• 2'b10: Initiator 使用 Target Sequence Context ( SC ) 资源。</li> <li>• 2'b11: Reserved。</li> </ul>	无此域段
INI_RC_ID	20	<p>Initiator Requester Context ID:</p> <ul style="list-style-type: none"> <li>• 若 INI_RC_Type 为 2'b00 或 2'b01, 此域段表示 Requester Context ID ( RCID ), Target 生成事务响应时, 把此域段带回给 Initiator, 以便 Initiator 用</li> </ul>	无此域段

域段名	位宽( bits )	完整 ATAH 域段描述	精简 ATAH 域段描述
		来索引正确的 SQ。 • 若 INI_RC_Type 为 2'b10, 此域段表示 SCID, Target 产生事务响应时, 从 SC 中取出 RCID, 填到 ATAH 中, 以便 Initiator 用来索引正确的 SQ。	

不同事务响应使用的包格式如下:

表 7-4 事务响应包格式

TAOpcode	事务操作	事务层包格式
0x11	Transaction_ACK	ATAH
0x12	Read_response	ATAH,[TAIDTAH],[OFSTTAH],Payload
0x13	Atomic_response	ATAH,[TAIDTAH],[Payload]

注: [\*]为可选包头, 是否携带见 7.4 节。

### 7.2.3 Memory Access Extended Transaction Header ( MAETAH )

MAETAH 携带内存访问中 Target 内存段 (见 8.2.1 节) 的地址信息、数据长度和 TokenID。MAETAH 包含完整格式以及精简格式。

完整 MAETAH 格式如下:

Byte0								Byte1								Byte2								Byte3										
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			
Address[63:32]																																		
Address[31:0]																																		
RSVD		TokenID[19:0]		RSVD						Length																								

图 7-6 完整 MAETAH 格式

精简 MAETAH 格式如下：

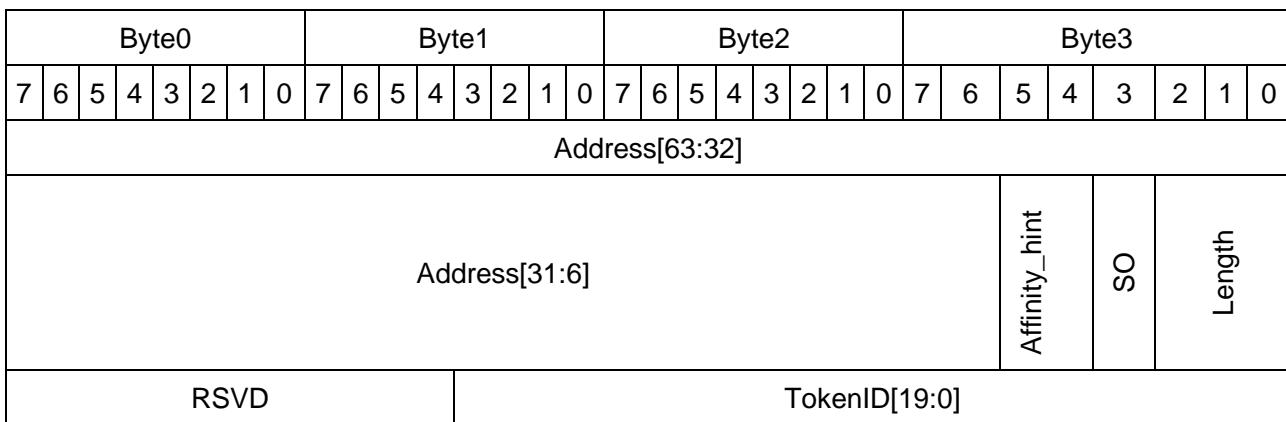


图 7-7 精简 MAETAH 格式

各域段描述如下：

表 7-5 MAETAH 域段

域段名	位宽 ( bits )	完整 MAETAH 域段描述	精简 MAETAH 域段描述
Address	64	待访问的 Target 内存起始地址。 对于 Atomic，要求 Atomic 操作数的地址是对齐的，例如，4 Bytes Atomic 事务的地址必须基于 4 Bytes 地址对齐。	同左，低 6 bits 为 0。
TokenID	20	TokenID，访问 Target 内存段的索引。	同左
Affinity_hint	2	无此域段	指示 Target 数据访问或处理的亲和性
SO	1	无此域段	保序执行标记 <ul style="list-style-type: none"> <li>1'b0: RO，会阻塞后续标记为 SO 的事务。</li> <li>1b'1: SO，需要等待前序标记为 SO 或 RO 的事务执行完才能执行。</li> </ul>
Length	32/3	访问 Target 内存的数据长度，如果事务被切片，该域段标识单个切片要访问的内存段的数据长度。	访问 Target 内存的数据长度，数据长度为 $64B \times (2^{length})$ ，当 length 为 0 时，访问数据长度为 64B，当 length 为 7 时，访问数据长度为 8192B。

### 7.2.4 Message Target Extended Transaction Header ( MTETAH )

MTETAH 用于描述 Target 处理消息事务的对象信息，即事务队列 RQ 信息（见 8.2.3 节），事务层中使用 TCID ( Target Context ID ) 作为消息事务处理对象的索引。携带该扩展头时，BTAH.MT\_EN 应置为 1。

MTETAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
Hint				RSVD		TGT_TC_Type	TGT_TC_ID																								

图 7-8 MTETAH 格式

各字段描述如下：

表 7-6 MTETAH 域段

域段名	位宽 ( bits )	MTETAH 域段描述
Hint	8	仅 TGT_TC_Type 为 Target Context ( TC ) Group 时生效。Target 收到包，根据此 Hint 把消息散列到不同 RQ，以实现负载均衡。
TGT_TC_Type	2	指示 Target Context 类型： <ul style="list-style-type: none"><li>• 2'b00 或 2'b01: TC。</li><li>• 2'b10: TC Group。</li><li>• 2'b11: Reserved。</li></ul>
TGT_TC_ID	20	指示 RQ 或 RQ group 的索引 ID，即 TCID/TC Group ID。

### 7.2.5 TokenValue Extended Transaction Header ( TVETAH )

TVETAH 携带 TokenValue，用于 Target 安全校验（见 8.2.4 节）。如果不启用访问权限校验，则无需携带该扩展头。携带该扩展头时，BTAH.TV\_EN 应置为 1。

对于 Send/Send\_with\_immediate 事务请求，该域段用于 Jetty/JFR/Jetty Group（见 8.2.2 节）的安全校验，Target 基于 MTETAH 的 TCID 索引到 TokenValue 进行安全校验。当 Payload 长度为 0 时，Jetty/JFR/Jetty Group 仍需进行安全校验。

对于 Read/Atomic/Write 事务请求，该域段用于 Target 内存段的安全校验，Target 基于 MAETAH 的 TokenID 索引到 TokenValue 进行安全校验。当 Payload 长度为 0 时，访问 Target 内存段时无需进行安全校验。

TVETAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
TokenValue[31:0]																															

图 7-9 TVETAH 格式

各字段描述如下：

表 7-7 TVETAH 域段

域段名	位宽( bits )	TVETAH 域段描述
TokenValue	32	用于 Target 安全校验。根据访问对象的不同，分为访问内存段的 TokenValue 和访问 Target Jetty/JFR/Jetty Group 使用的 TokenValue。

## 7.2.6 Task ID Extended Transaction Header ( TAIDETAH )

TAIDETAH 携带 Task ID 信息，用于 Initiator 索引到对应的事务。

TAIDETAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
RSVD																TAID															

图 7-10 TAIDETAH 格式

各字段描述如下：

表 7-8 TAIDETAH 域段

域段名	位宽 ( bits )	TAIDETAH 域段描述
TAID	16	任务 ID。Target 返回事务响应时携带此域，方便 Initiator 索引到对应的发送任务，来接收事务响应。

### 7.2.7 Offset Extended Transaction Header ( OFSTETAH )

OFSTETAH 携带偏移量信息，用于指示该包在事务层多个包内的偏移位置。

OFSTETAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
RSVD								Offset																							

图 7-11 OFSTETAH 格式

各域段描述如下：

表 7-9 OFSTETAH 域段

域段名	位宽( bits )	OFSTETAH 域段描述
Offset	24	<p>表示包在事务层多个包内的偏移量，以 1K Bytes 为粒度，首个包 offset 为 0，后续包 Offset 按 1KB 个数依次递增。例如，若首个包对应 nKB Payload，则第二个包 offset 为 n，依次类推。</p> <p>对于 Send 事务而言，Target 通过 Offset 信息确定该包在 Receive Queue Element ( RQE ) 中的偏移量。CTP 模式下，Send 事务长度限制在 1 个包以内，该域段 Reserved。</p> <p>对于 Read 事务而言，Initiator 通过 Offset 信息确定该 Read Response 包在 Send Queue Element ( SQE ) 中的偏移量。</p>

### 7.2.8 Immediate Extended Transaction Header ( IMMETAH )

IMMETAH 携带立即数信息，Target 会将立即数存放至 CQE 内上报给用户。Send\_with\_immediate 和 Write\_with\_immediate 应携带该扩展头。

IMMETAH 格式如下：

Byte0								Byte1								Byte2								Byte3							
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
Immediate_Data[63:32]												Immediate_Data[31:0]																			
Immediate_TokenValue												Msg_Length																			

图 7-12 IMMETAH 格式

各字段描述如下：

**表 7-10 IMMETAH 域段**

域段名	位宽( bits )	IMMETAH 域段描述
Immediate_Data	64	立即数， Target 产生完成通知时将立即数放在 CQE 内上报给用户。立即数长度固定为 8 Bytes。
Immediate_TokenValue	32	访问 Target RQ 时使用此域段与对应 Jetty 的 TokenValue 进行校验。仅 Write_with_immediate 事务会使用该域段，其余操作该域段 Reserved。
Msg_Length	32	指示 Write 事务的总长度。用于 Target 上报 CQE。仅 Write_with_immediate 事务会使用该域段，其余操作该域段 Reserved。

### 7.2.9 Notify Extended Transaction Header ( NTFETAH )

NTFETAH 携带 Notify 数据和该数据要写入的地址信息以及访问凭据。Write\_with\_notify 应携带该扩展头。

NTFETAH 格式如下：

Byte0								Byte1								Byte2								Byte3																																		
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																											
Address[63:32]																																																										
Address[31:0]																																																										
RSVD		Notify_TokenID[19:0]																									RSVD																															
Notify_TokenValue																																																										
Notify_Data[63:32]																																																										
Notify_Data[31:0]																																																										

**图 7-13 NTFETAH 格式**

各字段描述如下：

**表 7-11 NTFETAH 域段**

域段名	位宽 ( bits )	描述
Address	64	Notify 数据要写入的 Target 内存起始地址。
Notify_TokenID	20	TokenID， 访问 Notify 数据要写入的内存段时使用的索引。

域段名	位宽 ( bits )	描述
Notify_TokenValue	32	访问 Notify 数据要写入的内存段时使用的 TokenValue。 注 1: BTAH.TV_EN 为 1 时, 包头携带 2 个 TokenValue, TVETAH 中的 TokenValue 用于 MAETAH 的内存段权限校验, NTFETAH 中的 TokenValue 用于 Notify 写入的内存段的权限校验。 注 2: BTAH.TV_EN 为 0 时, NTFETAH.Notify_TokenValue 为 Reserved, 不使用。
Notify_Data	64	Notify 数据。有效数据固定 8 Bytes。

### 7.2.10 Byte Enable Extended Transaction Header ( BEETAH )

BEETAH 用于指示 Payload 的字节有效情况。Write\_with\_be、Writeback\_with\_be 应携带该扩展头。

BEETAH 格式如下 ( 以 8B 为例 ):

Byte0								Byte1								Byte2								Byte3							
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
BE[63:32]																															
BE[31:0]																															

图 7-14 BEETAH 格式 ( 以 8B 为例 )

各域段描述如下:

表 7-12 BEETAH 域段

域段名	位宽 ( bits )	描述
BE	64~1024	Byte Enable。BE[n]表示包的网络传输字节序中的 Payload 第 n 个字节是否有效, 为 1 时表示有效, 否则无效。 BE 支持 64/128/256/512/1024 bits, 分别对应 Write_with_be、Writeback_with_be 事务的 64/128/256/512/1024 Bytes Payload 长度。

### 7.2.11 User Defined Extended Transaction Header ( UDETAH )

UDETAH 携带的具体内容由厂商自定义, 其长度为 4 Bytes, 由 BTAH.UD\_Flag 域段指示是否携带 UDETAH。

Byte0								Byte1								Byte2								Byte3							
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
UD[31:0]																															

图 7-15 UDETAH 格式

### 7.2.12 Management Extended Transaction Header ( MGMTETAH )

Management 事务应携带该扩展头。包格式见 10.4.3.3 节。

## 7.3 事务服务

### 7.3.1 事务可靠性

#### 7.3.1.1 可靠事务

可靠事务服务下，Initiator 发送事务请求后，Target 需要针对该事务请求返回事务响应，包含成功、异常或者其它执行状态。

可靠事务服务交互流程如下图所示：

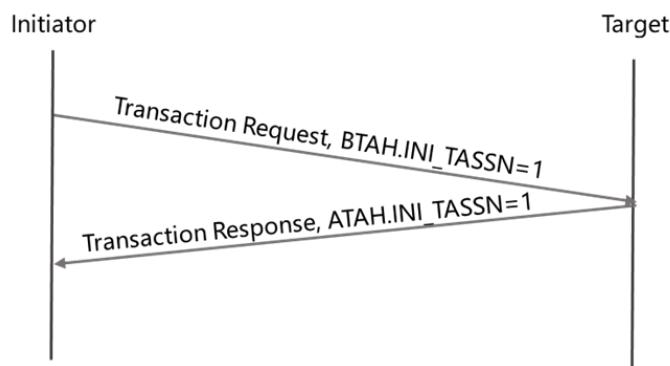


图 7-16 可靠事务服务交互流程

通常，一次事务操作包含一次事务请求以及一次事务响应，但事务层应支持事务切片功能，即一个事务被切分成多个事务切片，每个事务切片对应单独的 TASSN。每个事务切片都对应一次事务请求以及一次事务响应。Initiator 只有当收齐所有的事务切片响应后，才认为整个事务完成。事务切片应遵循如下约束：

1. 当传输层采用 RTP 模式时，每个事务切片可分散到不同的 TP Channel 上，防止大事务占据 TP Channel 时间过长形成阻塞，切片粒度可配置。比如，一个 1MB 的写内存事务，可以切分成多个 64KB 的事务切片。是否支持事务切片见具体事务处理，见 7.4 节。
2. 当传输层采用 CTP 或 TP Bypass 模式时，一个事务切片对应一个包，即每个包都需要一个单独的 TASSN。

事务响应包括 TAACK、Read Response 以及 Atomic Response。当需要返回正常 TAACK，且事务层采用完整包格式时，事务层支持一对 Initiator 和 Target 之间连续 TASSN 的 TAACK 后向聚合，即多个 TAACK 通过一个 TAACK 返回给发送端，当 Initiator 收到一个聚合的 TAACK 时可认为之后已经聚合的 TAACK 均收到。例如 TAACKTASSN 为 N，且 ATAH.RSPST 为 3' b000 或 3' b101，ATAH.RSPINFO

为 9, Initiator 收到 TAACK 时, 认为收到了 N~N+9, 共 10 个事务响应。针对异常 TAACK, 事务层不支持 TAACK 聚合。

通常情况下, Target 需要返回事务响应, 但存在如下特殊情况:

1. Target 在一些场景下可以通过传输层应答包随路携带事务响应状态, 包括:
  - (1) 当传输层采用 RTP, 事务服务模式为 ROL (见 7.3.3.4 节) 时, 可通过传输层应答携带事务执行状态, 无单独的 TAACK。该场景 TAACK 不支持聚合。
  - (2) 当传输层采用 RTP, 事务服务模式为 ROI、ROT 以及 ROL (见 7.3.3.2-7.3.3.4) 时, 如果事务层资源不足无法返回事务响应(包括 TAACK、Read Response 以及 Atomic Response), 可通过传输层应答携带事务层执行状态。该场景事务响应不支持聚合。
2. 当传输层采用 CTP 时, 并且事务服务模式为 ROL (见 7.3.3.4 节) 时, 用户可通过在请求包中设定 BTAH.No\_TAACK 域段为 1, 使 Target 不返回 TAACK, 以降低响应包数量。注: 此方式只适用于网络可靠状态极佳的场景。

事务层应支持采用如下两种机制实现可靠事务服务:

1. 下层协议应保证事务层包传输的可靠性, 即保证事务层包在 Initiator 和 Target 之间的无损传输。
2. 事务层及上层保证事务执行的可靠性, 包括:
  - (1) 事务层针对可处理的异常应进行事务重传, 包括 RNR 以及 Page Fault (异常状态通过 ATAH 携带)。事务重传的流程图如下图所示:

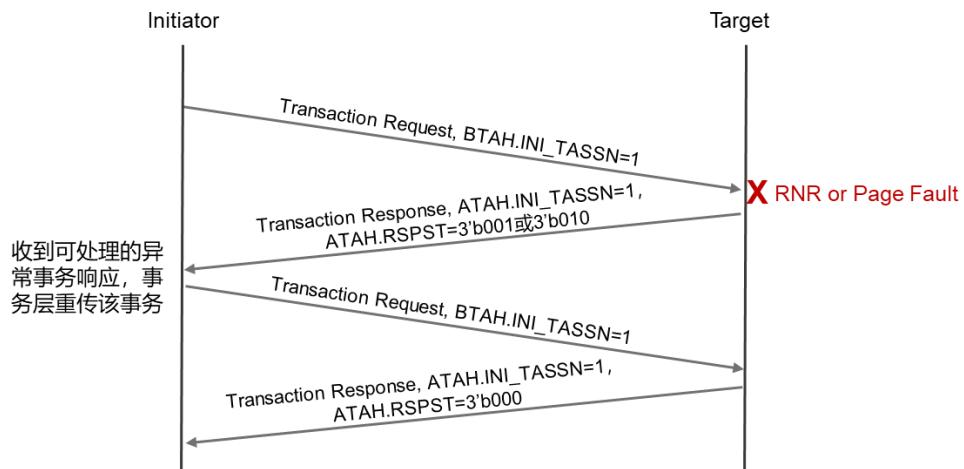


图 7-17 可靠事务服务重传流程

- (2) 针对不可处理的异常, 如内存长度异常, 权限校验不通过等, 事务层不支持重传。Initiator 将异常信息递交给上层, 由上层完成异常处理, 如清除异常事务、事务请求重新下发等。事务异常情况列表见 10.6.2 节。

值得注意的是, 如果事务层重传可能导致数据不一致, 事务层不应重传异常事务。例如, 针对 Atomic 事务, 如果处理 Atomic Response 时出现 RNR 或 Page Fault, 不能重传 Atomic 事务, 事务出现异常后直接

上报异常。但处理 Atomic Request 时如果出现 RNR 或 Page Fault，重传 Atomic Request 不会出现数据不一致，可以重传。

示例：Initiator 发送 Atomic\_fetch\_add request，Target 收到后将内存地址内的数据 X 更新为 1，同时将 X 初始值 0 返回给 Initiator，Target 返回的 atomic response 更新到 Initiator 内存出现 Page Fault 异常。如果 Initiator 重传 request，则 Target 收到重传事务请求后，会将 X 更新为 2，同时将 X=1 作为响应返回给 Initiator，导致业务流程异常。

### 7.3.1.2 不可靠事务

不可靠事务服务下，Initiator 发送事务请求后，即认为事务操作完成，上报完成信息，不需要 Target 返回事务响应。不可靠事务服务传输交互流程如下图所示：



图 7-18 不可靠事务服务交互流程

不可靠事务服务下，事务层不要求下层协议有可靠传输能力，并且事务层也不要求提供事务重传的能力，针对 Initiator 处理异常，Initiator 会将异常信息上报给上层，由上层完成进一步处理，事务异常情况列表见 10.6.2 节。

### 7.3.2 事务序

#### 7.3.2.1 事务序概念

事务序是指同一事务队列中事务间的顺序关系，分为事务执行序和事务完成序。同一事务队列（如 SQ）的不同事务可以指定不同的事务序关系，如无顺序要求、有顺序要求等，不同队列间的事务没有事务序关系。

注：是否需要保序由上层指定，事务层只提供保序的机制。

#### 7.3.2.2 事务执行序

事务执行序是指事务请求在 Target 的执行顺序，通过事务执行序标记标识。可分为几类：

1. NO: No Order，表示该事务与其它事务之间没有保序要求，可任意顺序执行。

2. RO: Relaxed Order, 表示该事务的执行不依赖于其它事务, 可乱序执行, 并且会在执行完成之前阻塞后续标记为 SO 的事务, 即标记为 RO 和 SO 的事务有保序要求, 标记为 SO 的事务不能在标记为 RO 的前序事务之前执行。
3. SO: Strong Order, 表示该事务须保序执行, 须等它前序有 RO 或 SO 标记的事务执行完成后才能执行。

事务执行序可以在 Initiator、Target 或依赖于其它方式实现, 事务层提供多种保序机制实现事务间保序, 见 7.3.3 节。

Initiator 除可利用这些保序标记在事务层实现保序外, 也可通过其它功能实现保序, 如 Fence、Barrier 等, 协议不做规定。两种保序方式可配合使用。

事务层保序案例:

表 7-13 是一个事务队列, 序号从 0 到 4, 其中事务 0 标识为 RO, 事务 1 标识为 SO, 事务 2 标识为 NO, 事务 3 标识为 NO ( FENCE ), 事务 4 标识为 NO。Initiator 从事务 0 开始发送事务。

注: Fence 为应用或处理器实现的保序机制, 当一个事务使能了 Fence, 表示该事务必须等待其前面所有 Read 和 Atomic 类型的事务都执行完成并且收到响应, 才能发送出去, 否则该事务需一直在队列内等待, 且会阻塞后序事务的发送。

**表 7-13 事务执行序示例**

4: Read NO	3: Write NO ( FENCE )	2: Read NO	1: Write SO	0: Write RO
---------------	--------------------------	---------------	----------------	----------------

1. 事务 0 标识了 RO, 直接发送至 Target。
2. 事务 1 标识了 SO, 按照 SO 的规则, 事务 1 必须等事务 0 完成后才能执行。这时根据不同的保序模式有两种情况:
  - (1) Initiator 保序, Initiator 不能直接发送事务 1 对应的包。由于存在乱序可能, 事务 1 虽然是后发送, 却有可能比事务 0 的包先到达, 因此需要等到收齐事务 0 执行完成的事务响应包后才能发送事务 1。
  - (2) 非 Initiator 保序, Initiator 可直接发送事务 1 对应的包。当 Target 收到带有 SO 标记的事务 1 的包时, 需等待事务 0 执行完毕后才执行。
3. 事务 2 标识了 NO, 没有执行序的要求, 也不会被 SO 阻塞, 可以超越事务 1 执行。
4. 事务 3 标识了 NO ( FENCE ), 应用需等待前序 Read 都执行完毕才可以发送该事务。此时 事务 3 会阻塞事务 4 的发送 (因为事务 3 和事务 4 处于相同事务队列中, 事务 3 会阻塞事务 4)。
5. 事务 4 标识了 NO, 没有执行序的要求, 需等待事务 3 发送后才可发送至 Target。事务 4 可能会先于事务 3 完成。

### 7.3.2.3 事务完成序

事务完成序是指事务在执行完成后产生完成通知的顺序，包含发送完成序和接收完成序。仅应用于需要产生完成通知 CQE 的事务中。

注 1：事务服务模式划分只与事务执行序有关，与事务完成序无关，但上层仍可指定事务完成序实现完成通知的保序。

注 2：Initiator 是否产生 CQE 由用户下发事务请求时指定；Target 消耗 RQE 时应产生 CQE。

发送完成序是在 Initiator 产生完成通知的顺序，支持两种：

1. 保序完成：Initiator 事务完成通知产生顺序与事务下发顺序保持一致，即先下发的事务一定先产生完成通知。
2. 乱序完成：Initiator 事务完成通知产生顺序与事务下发顺序可以不一致，即后下发的事务可能先产生完成通知。

接收完成序是在 Target 产生完成通知的顺序，也支持两种：

1. 保序完成：Target 事务完成通知产生顺序与事务发送顺序保持一致，先发送的事务一定先产生完成通知。
2. 乱序完成：Target 事务完成通知产生顺序与事务发送顺序可以不一致，先处理完成的事务先产生完成通知。

当 Initiator 要求 Target 按序产生完成通知时，需要将事务完成序标记携带在事务层包头中，发送到 Target。

### 7.3.3 事务服务模式

#### 7.3.3.1 概述

根据事务层是否提供可靠性，以及是否保证事务执行序，事务服务模式可分为 ROI、ROT、ROL、UNO 四类。当存在多种事务服务模式时，由上层编程模型根据业务需求指定。

四种事务服务模式特点如下：

1. ROI 保证事务可靠执行，针对有执行序要求的事务，Initiator 需要等待所有有依赖关系的前序事务响应收齐后才能发送新事务。
2. ROT 保证事务可靠执行，针对有执行序要求的事务，Target 维护事务序，可以节省一个 RTT 时间，但需要额外的资源保证按序执行。
3. ROL 保证事务可靠执行，利用下层协议特点实现事务间保序。
4. UNO 不保证事务可靠执行，也无事务序的保证。

当有可靠要求的事务间无保序要求时，所有的事务均可以乱序传输与执行，ROI、ROT、ROL 模式无差异。

### 7.3.3.2 ROI 模式

ROI 模式提供可靠的、Initiator 针对有执行序要求的事务保证执行顺序的服务模式，该模式下事务层的包在路径上可乱序传输。由于事务层包在传输和执行可能存在乱序，Initiator 需要等到有保序关系的事务收到响应且正确执行完成后，才能发送新的事务。

由于 ROI 模式下允许事务层包乱序，即可以多路径传输。事务层与下层协议配合的关系可采用如下方式：

1. 传输层采用 RTP 时，事务层应为每个事务指定 TPG 或者不同的 TP Channel，且网络层可设置为逐包或逐流负载均衡。
2. 传输层采用 CTP 或 TP Bypass 时，网络层推荐设置为逐包负载均衡。

假如事务 1, 2, 3 的保序标记分别为 NO、RO、SO，ROI 模式下事务交互流程如下图所示。

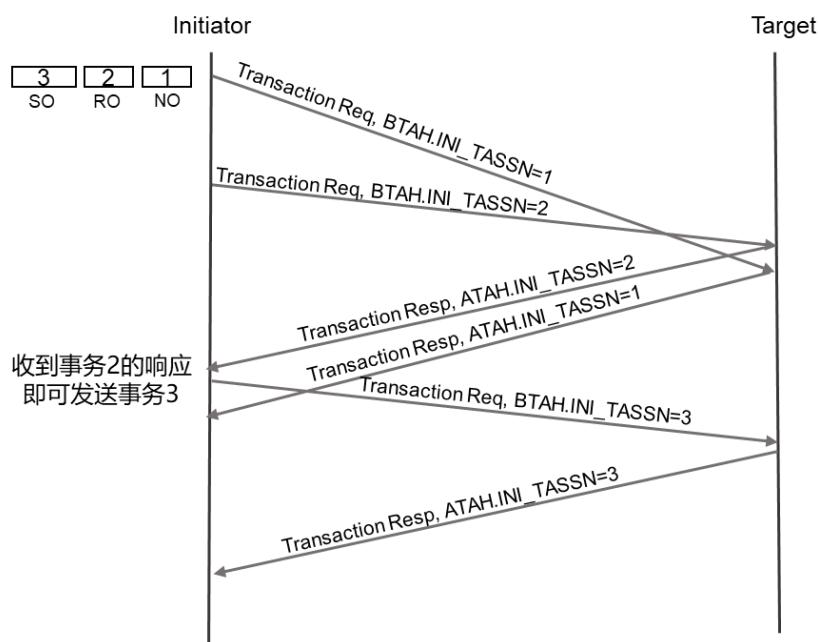


图 7-19 ROI 模式交互流程

### 7.3.3.3 ROT 模式

ROT 模式提供可靠的、Target 针对有执行序要求的事务保证执行顺序的服务模式，该模式下事务层的包在路径上可乱序传输。相对于 ROI 模式，事务保序功能实现在 Target，Initiator 可将有保序关系的事务都发送到 Target，无需等待有保序关系的所有事务响应，可节省等待时间。

假如事务 1, 2, 3 的保序标记分别为 NO、RO、SO，ROT 模式下事务交互流程如下图所示。

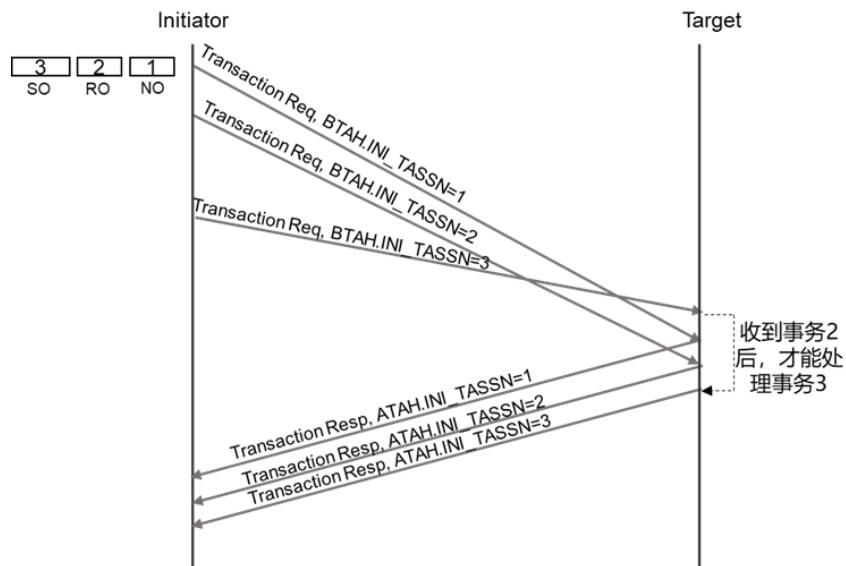


图 7-20 ROT 模式交互流程

为判断事务是否能执行，接收端需要单独的资源（Sequence Context，SC）维护事务执行状态，如 TASSN、SO/RO/NO 标记等。当收到乱序的事务，Target 如何做保序处理，协议不做规定，可采用如下两种方式：

1. Target 收到事务后，如果 TASSN 不是顺序递增的，则丢弃乱序事务，发送异常 TAACK，通知 Initiator 重传。该方法实现简单，但存在大量的重传。
2. 收到带有 SO 标记的事务后，如果 TASSN 乱序，则把该事务缓存下来，直到前序带有 SO 以及 RO 的事务都执行完，才能执行该事务。该方法重传少，但需要更多的缓存。

为节省资源开销，Initiator 需要先申请 SC 资源，如果申请到，Target 可进行保序处理，否则接收端无保序功能，回退到其它保序模式。资源申请分为静态配置以及动态申请两类。

当使用动态申请 SC 资源方式时，申请流程以及使用流程如下：

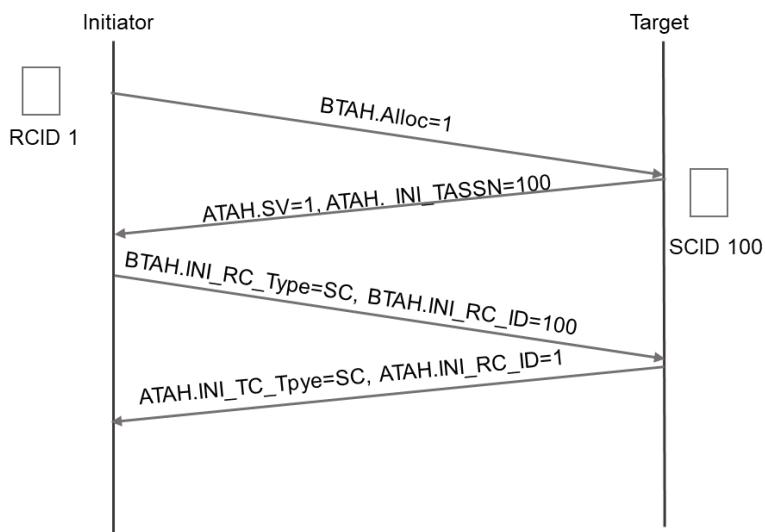


图 7-21 ROT 模式动态申请资源

1. 申请 SC 资源时，事务请求包中的 BTAH.Alloc 应置为 1。
2. Target 反馈的事务响应中，ATAH.SV 字段应设置为 0 或 1。当设置为 1 时，代表申请到 SC 资源，ATAH.INI\_TASSN 代表申请到的 SCID；若设置为 0，代表未申请到资源，接收端不提供保序能力。如果申请到资源，Target 将会把 SCID 和 Initiator 事务发送上下文（RCID）建立映射关系。
3. 如果申请到 SCID，再次发送事务操作时，应将 BTAH.INI\_RC\_Type 设置为 SC，且事务层包头中的 BTAH.INI\_RC\_ID 应设置为申请到 SCID。
4. Target 再次反馈事务响应时，ATAH.INI\_RC\_ID 应设置为与 SCID 对应的 RCID。

注：如果静态分配资源时，步骤 1,2 可省略。其余步骤相同。

ROT 模式允许事务层包在线路上乱序传输，与传输层或网络层的配合关系和 ROI 模式相同。

#### 7.3.3.4 ROL 模式

ROL 模式提供可靠的、Target 依赖于下层协议保证事务执行序的服务模式，该模式下事务层的包是否可乱序传输取决于事务层之下是否有保序的能力。相比于 ROI 和 ROT 模式，事务保序由事务层之下的协议层完成。

当事务间存在执行序要求时，下层协议为保证按序执行，应保证按顺序接收事务层包，当采用不同的下层协议时，有不同的要求。ROL 模式与下层协议配合关系可遵循如下方式：

1. 传输层采用 RTP 时，需要保序的事务应指定相同的 TP Channel，且网络层可采用逐包或逐流负载均衡。当采用逐包负载均衡时，事务层包可乱序传输，由接收端传输层完成事务层包按序向上递交。
2. 传输层采用 CTP 或 TP Bypass 传输层时，网络层应采用逐流负载均衡，即采用单路径传输。

除此之外，由于 Target 片上总线可能存在乱序，为保证按序执行，下层协议需要保证有保序关系的前序事务已经被处理后才能递交新的事务。值得注意的是，如果下层协议不具备该能力，事务层要配合完成该功能。

假如事务 1, 2, 3 的保序标记分别为 NO、RO、SO，ROL 模式下事务交互流程如下图所示。

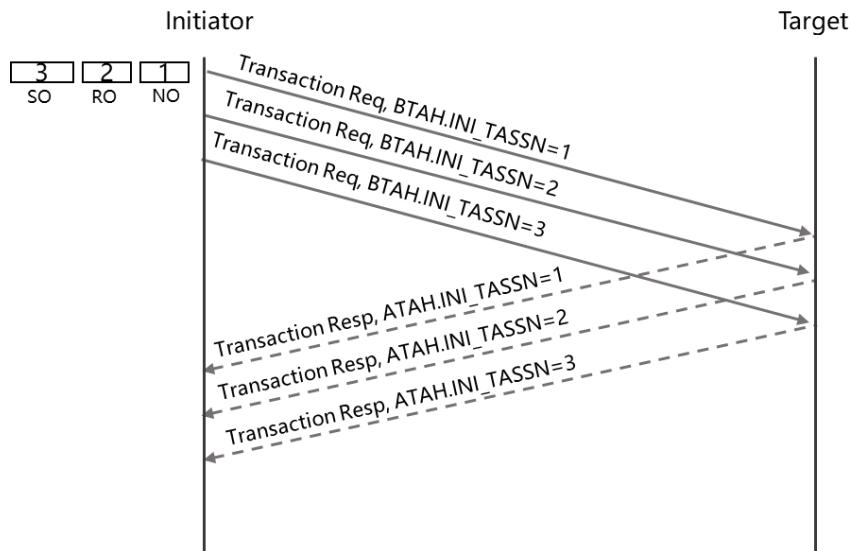


图 7-22 ROL 模式交互流程

### 7.3.3.5 UNO 模式

UNO 模式提供不可靠的、不保序的事务服务模式，该模式下事务层的包允许在网络上乱序传输。

当使用 UNO 模式时，事务数据应不超过一个传输层 MTU 大小。

UNO 模式下可使用 UTP、CTP 或 TP Bypass 模式。

### 7.3.4 与下层配合关系

事务层提供可靠事务服务时，分为包传输可靠性以及事务执行可靠性。包传输可靠性依赖下层协议保证。

事务层可根据上层需要提供事务保序能力。当采用 Initiator 保序 ( ROI 模式 ) 或 Target 保序 ( ROT 模式 ) 时，事务层包可多路径传输，以充分利用网络带宽。实现多路径的方式，可以在传输层指定 TPG 或多 TP Channel，也可以在网络层指定负载均衡模式，或者二者搭配使用，详细描述参考传输层以及网络层。

当事务保序能力卸载到下层处理 ( ROL 模式 ) 时，可分为两种情况：

1. 当传输层采用 RTP 时，可利用 TP Channel 按序处理能力实现事务间保序，即需要保序的事务可通过指定相同的 TP Channel 实现保序。此情况下，事务层包仍可以在路径中乱序传输，由 Target 传输层恢复顺序。
2. 当传输层采用 CTP 或者 TP Bypass 时，传输层无保序能力，针对有保序要求的事务，需要通过单路径传输。此时，应配置多路径模式为逐流均衡。

当事务有保序要求时，事务服务模式与传输层以及网络层、数据链路层的组合关系如下表所示。

表 7-14 事务层和下层配合关系（针对有执行序要求的事务）

模式	传输层	网络层	数据链路层
ROI、ROT	RTP：指定 TPG 或多 TP Channel	RT[0]: 可设置为 0 或 1	推荐可靠

模式	传输层	网络层	数据链路层
	CTP/TP Bypass: 无需指定	RT[0]: 推荐设置为 1	要求可靠
	UTP: 不支持	/	/
ROL	RTP: 指定单 TP Channel	RT[0]: 可设置为 0 或 1	推荐可靠
	CTP/TP Bypass: 无需指定	RT[0]: 应设置为 0	要求可靠
	UTP: 不支持	/	/
UNO	CTP/TP Bypass: 无需指定	RT[0]: 设置为 0 或 1	无要求
	UTP: 无需指定	RT[0]: 设置为 0 或 1	无要求
	RTP: 不推荐使用	/	/

当事务无执行序要求时，事务默认可以多路径传输以及乱序执行，事务服务模式与传输层以及网络层、数据链路层的组合关系如下表所示。

表 7-15 事务层和下层配合关系（针对无执行序要求的事务）

模式	传输层	网络层	数据链路层
ROI、ROT、ROL	RTP: 指定 TPG 或多 TP Channel	RT[0]: 可设置为 0 或 1	推荐可靠
	CTP/TP Bypass: 无需指定	RT[0]: 推荐设置为 1	要求可靠
	UTP: 不支持	/	/
UNO	CTP/TP Bypass: 无需指定	RT[0]: 设置为 0 或 1	无要求
	UTP: 无 TP Channel	RT[0]: 设置为 0 或 1	无要求
	RTP: 不推荐使用	/	/

## 7.4 事务类型

### 7.4.1 概述

根据事务操作功能不同，事务层支持内存事务、消息事务、维护事务以及管理事务四类。

1. Initiator 使用内存事务可直接访问对端内存，如 Write、Read、Atomic 等，见 7.4.2。
2. Initiator 使用消息事务向 Target 传递消息，由 Target 进行消息处理，如 Send 等，见 7.4.3。
3. Initiator 通过维护事务更新对端运行过程中的状态，如缓存状态等，见 7.4.4。
4. Initiator 通过管理事务实现设备管理、故障上报等，见 7.4.5。

每种事务类型包含多种事务子类型，不同事务子类型的 TAOpcode 不同，事务层针对每种事务类型定义交互流程以及包格式。事务层使用不同事务服务模式，以及传输层模式时，事务操作的处理流程存在差异化，如是否需要等待响应才能发送事务请求，是否允许事务多路径传输等，详细差异点见 7.4.2-7.4.5。除已定义的事务类型外，UB 支持扩展更多事务子类型，以实现不同场景下更极致通信效率。

## 7.4.2 内存事务

### 7.4.2.1 Write 事务

Initiator 使用 Write 事务将本地数据写入 Target 的指定内存空间。Write 事务支持使用 ROI、ROT、ROL 事务服务模式。

Write 事务流程图如下所示：

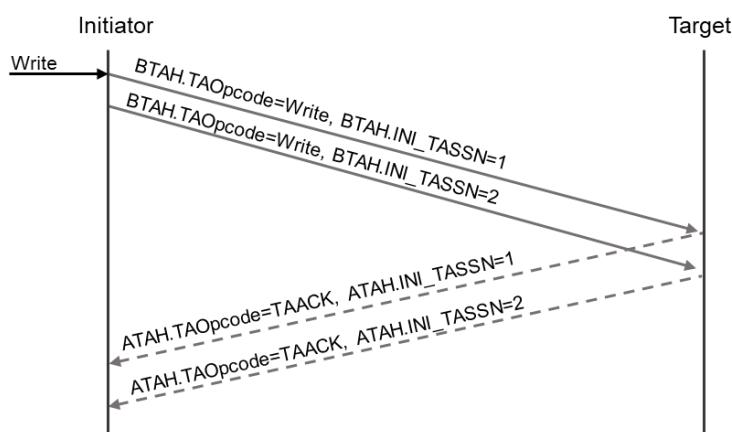


图 7-23 Write 事务处理流程

1. Initiator 发起 Write 事务到 Target。单个 Write 事务可以使用多个包发送，包格式如下，详细信息见 7.2。



其中：

- BTAH.TAOpcde：应设置为 0x3；
- BTAH.INI\_TASSN：事务请求编号，Target 返回 TAACK 时需要携带，Initiator 根据该字段识别哪个事务请求得到响应；

注：当传输层采用 RTP 时，Write 事务支持事务切片，事务切片大小可超过一个包；当传输层采用 CTP 或 TP Bypass 时，Write 事务支持切片，一个切片对应一个包。

- BTAH.UD\_Flag：可设置为 0 或 1；

- BTAH.TV\_EN: 可设置为 0 或 1;
  - MAETAH.Address: Target 被访问的内存地址, 当 Write 事务数据长度超过一个传输层 MTU 时, 每个包的 MAETAH.Address 应为上一个包的访问地址加传输层 MTU;
  - MAETAH.Length: Target 被访问的内存长度;
  - MAETAH.SO: 可设置为 0 或 1, 仅精简包格式下有效;
  - 当使用完整包格式时, 包头中应额外包含如下信息:
    - (1) BTAH.ODR: 无保序要求时, 应设置为 3'b000, 有保序要求时, BTAH.ORD 应设置为 3'b001 或 3'b010;
    - (2) BTAH.MT\_EN: 应设置为 0;
    - (3) BTAH.INI\_RC\_Type: 发起事务请求的 context 类型, 仅当事务服务模式为 ROT 且申请到 SCID 后, 才可设置为 2'b10, 否则都应设置为 2'b00 或 2'b01;
    - (4) BTAH.INI\_RC\_ID: Requester Context ID, 用于标记事务请求发起的上下文信息, 返回 TAACK 时, 需要携带该字段。
2. Target 收到 Write 包, 结合内存管理单元完成事务执行后返回 TAACK, 若事务执行过程中出现异常, 需要将异常信息携带在 TAACK 中返回给 Initiator。特殊情况不需要返回, 见 7.3.1.1 节。TAACK 包头格式如下, 详细信息见 7.2 节。

ATAH

其中,

- ATAH.TAOpcode: 应设置为 0x11;
  - ATAH.INI\_TASSN: 应与 Write 包中的INI\_TASSN 保持一致;
  - ATAH.Status: 可设置为 0 或 1, 仅在精简包格式中有效;
  - 当使用完整包格式时, 包头中应额外包含如下信息:
    - (1) ATAH.INI\_RC\_Type: 应与 Write 包中的 BTAH.INI\_RC\_Type 字段相同;
    - (2) ATAH.INI\_RC\_ID: 应与 Write 包中的 BTAH.INI\_RC\_ID 字段相同;
    - (3) ATAH.RSPST: 事务完成状态, 包括各种异常信息。
3. Initiator 收到 TAACK 后, 如事务响应正常, 通知上层处理, 若返回的 TAACK 中 ATAH.Status 或 ATAH.RSPST 显示事务处理异常, 进入异常处理流程, 见 7.3.1.1。

#### 7.4.2.1.2 Write\_with\_notify

Initiator 使用 Write\_with\_notify 事务将本地数据写入 Target 的指定内存空间, 并且在最后一个包中携带 Notify 数据。Notify 数据只有事务中所有的 Write 包被处理完才能被处理, 并且 Notify 数据需要写入和 Write 数据不同的指定内存中。Write\_with\_notify 事务可使用 ROI、ROT、ROL 事务服务模式。

由于 Notify 数据与同一事务的 Write 数据有保序要求, 事务层使用不同事务服务模式时, 事务处理流程存在差异。与下层的配合关系见表 7-14。

当事务服务模式采用 ROI 时，Write\_with\_notify 的处理流程图如下：

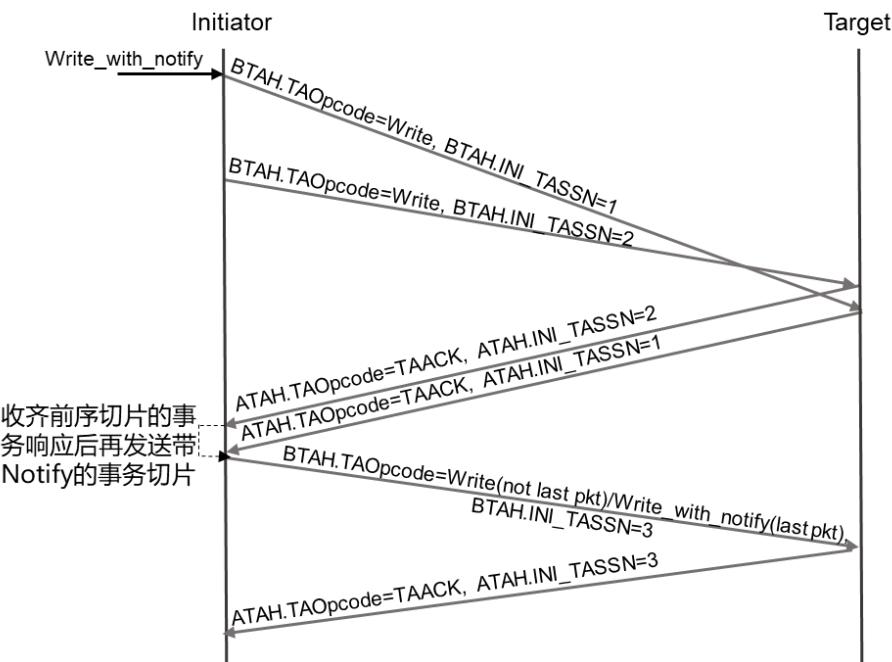


图 7-24 Write\_with\_notify 事务处理流程 (ROI 模式)

1. Initiator 发起 Write\_with\_notify 事务。其中 Write 部分的包格式以及域段设置同 Write，见 7.4.2.1.1。
2. Target 收到 Write 包后，更新内存后需要返回 TAACK，如果事务执行过程中出现异常，将异常信息携带在 TAACK 中。TAACK 包格式以及要求同 Write 事务，见 7.4.2.1.1。
3. Initiator 收到所有 TAACK 后，发送最后一个事务切片，切片中应携带 Notify 数据。如果收到的 TAACK 中 ATAH.Status 或 ATAH.RSPST 显示事务处理异常，进入异常处理流程，见 7.3.1.1。最后一个包的包格式如下，详细信息见 7.2。



其中，

- BTAH.TAOpcode：应设置为 0x5；
  - BTAH.UD\_Flag：可设置为 0 或 1；
  - BTAH.MT\_EN：应设置为 0；
  - 应包括 NTFETAH 头，包头中应包含 Notify 待写入的首地址、地址段对应的访问凭据以及 Notify 数据。
4. Target 收到 Notify 数据，返回 TAACK，TAACK 包格式同步步骤 2。
  5. Initiator 收到 TAACK 后，事务操作结束，若返回的 TAACK 中 ATAH.Status 或 ATAH.RSPST 显示事务处理异常，进入异常处理流程，见 7.3.1.1。

当事务服务模式采用 ROT 时， write\_with\_notify 的处理流程图如下：

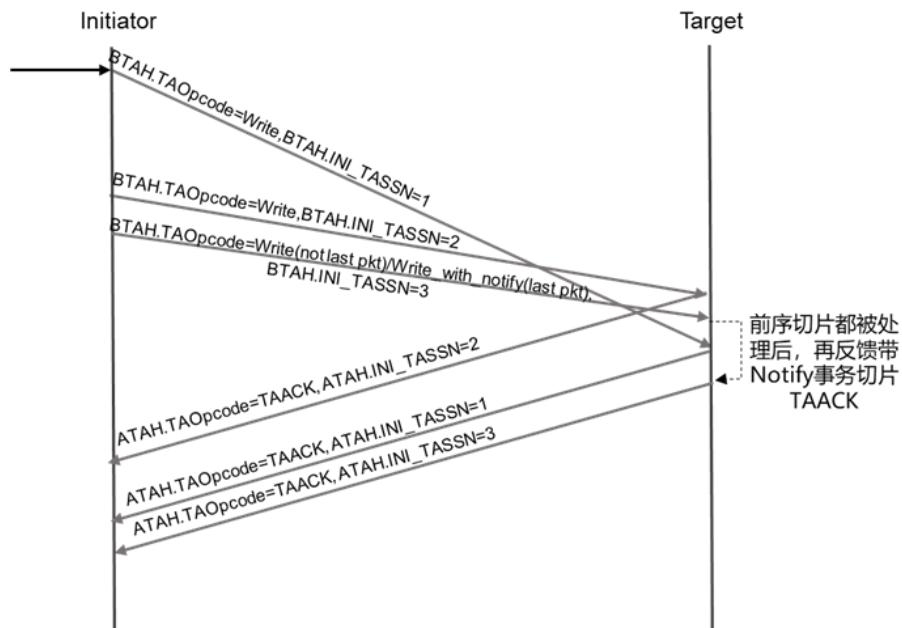


图 7-25 Write\_with\_notify 事务处理流程 (ROT 模式)

和 ROI 模式相比，事务操作处理流程的主要区别：ROT 模式下 Target 事务层具备保序能力，Notify 数据可和 Write 数据共同发往 Target，无需等待 Write 数据的 TAACK 都收到。即使先收到 Notify 数据，也要确保前面的 Write 数据都收到并写入内存后才可以写入 Notify 数据。

当事务服务模式采用 ROL 时， write\_with\_notify 的处理流程图如下：

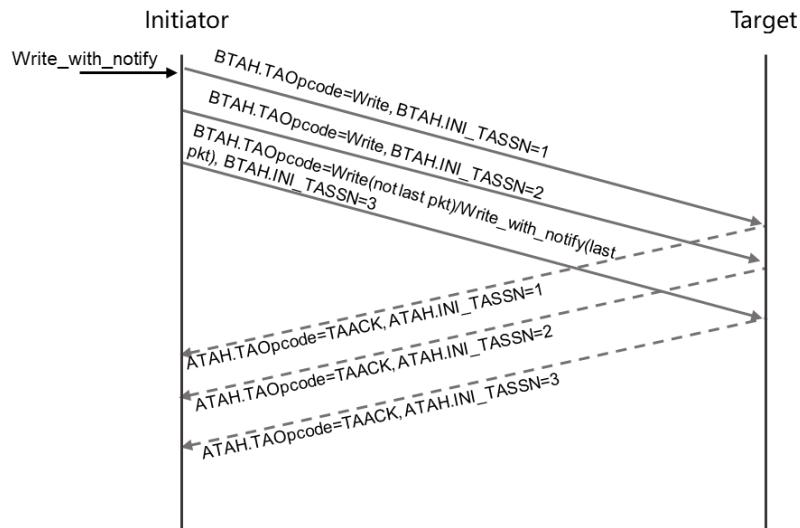


图 7-26 Write\_with\_notify 事务处理流程 (ROL 模式)

和 ROI 模式相比，事务操作处理流程的主要区别：

1. ROL 模式下要求下层协议具备保序能力，Notify 数据可和 Write 数据共同发往 Target，保序方式可见 7.3.3.4；

2. Target 收到事务层包后，当传输层采用 RTP 时，可通过传输层应答携带 TAACK，当采用 CTP 时，上层可选择是否需要返回 TAACK。

#### 7.4.2.1.3 Write\_with\_be

Initiator 使用 Write\_with\_be 事务将本地数据写入 Target 指定内存空间，同时携带 Byte Enable( BE ) 字段，指定 Payload 中哪些字节是有效位，仅将有效字节写入 Target 内存。

Write\_with\_be 事务仅包含一个包，BE 支持 64/128/256/512/1024 bits，分别对应 write\_with\_be 操作的 64/128/256/512/1024 Bytes Payload 长度。Write\_with\_be 事务可使用 ROI、ROT、ROL 事务服务模式。

Write\_with\_be 事务处理流程同 Write 事务相同（见 7.4.2.1.1）。

Write\_with\_be 包格式如下，详细信息见 7.2。



其中，

1. BTAH.TAOpcode：应设置为 0x14；
2. BTAH.UD\_Flag：可设置为 0 或 1；
3. BEETAH：应携带 BE 字段。通常情况，Payload 长度不应超过 BEETAH 字段位宽，但当 Payload 长度超过 BEETAH 字段位宽时，默认超过 BE 范围的 Payload 均有效。

#### 7.4.2.1.4 Writeback

Initiator 使用 Writeback 事务将本地 Cache 内数据写入 Target 指定内存空间。Writeback 事务可使用 ROI、ROL 事务服务模式。

Writeback 事务仅包含一个包，事务处理流程同 Write 事务相同。Writeback 包格式如下，详细信息见 7.2。



其中，

1. BTAH.TAOpcode：应设置为 0x17；
2. BTAH.UD\_Flag：可设置为 0 或 1；
3. BTAH.TV\_EN：可设置为 0 或 1。

和 Write 事务相比，Writeback 处理要求无阻塞执行，即不能被普通的读写操作阻塞，否则存在死锁风险。

#### 7.4.2.1.5 Writeback\_with\_be

Initiator 使用 Writeback\_with\_be 事务将本地 Cache 内数据写入 Target 指定内存空间，同时携带 BE 字段，指定 Payload 中哪些字节是有效位，仅将有效字节写入 Target 内存。

`Writeback_with_be` 事务仅包含一个包，BE 支持 64/128/256/512/1024 bits，分别对应 `Writeback_with_be` 事务的 64/128/256/512/1024 Bytes Payload 长度。`Writeback_with_be` 事务可使用 ROI、ROL 事务服务模式。

事务处理流程同 Write 事务处理流程相同。

`Writeback_with_be` 包格式如下，详细信息见 7.2。



其中，

1. BTAH.TAOpcode：应设置为 0x18；
2. BTAH.UD\_Flag：可设置为 0 或 1；
3. BTAH.TV\_EN：可设置为 0 或 1；
4. 包格式中应包含 BEETAH 头，指示有效字节。

`Writeback_with_be` 处理遵循的原则和 Writeback 相同。

#### 7.4.2.2 Read 事务

Initiator 使用 Read 事务将 Target 指定内存空间的数据读取到本地内存空间，Read 事务支持 ROI、ROT、ROL 事务服务模式，并且仅支持针对 Request 的事务切片。

Read 事务的处理流程图如下所示：

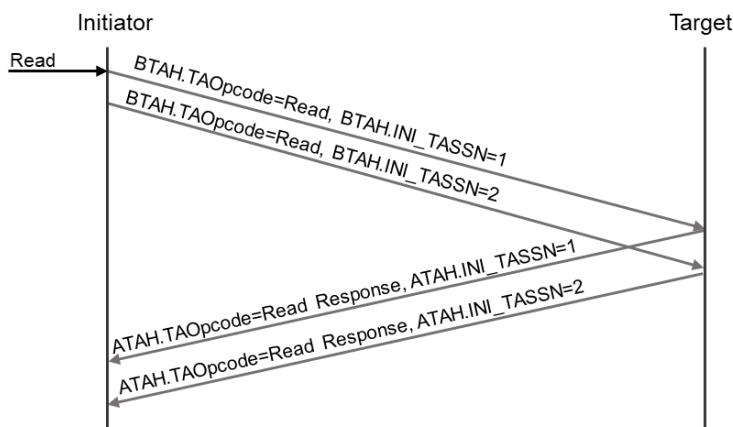
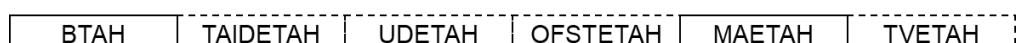


图 7-27 Read 事务处理流程

1. Initiator 发起 Read Request 到 Target。事务层请求包格式如下，详细信息见 7.2。



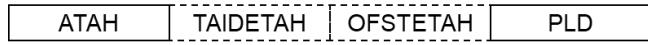
其中，

- BTAH.TAOpcode：应设置为 0x6；
- BTAH.INI\_TASSN：事务操作编号，Target 返回 Read Response 时，需要携带，Initiator 根据该域段识别哪个事务请求得到响应；

注：当传输层采用 RTP 时，Read 事务支持 Request 切片，该域段为每个 Request 切片的编号，一个切片对应的响应可超过一个包；当传输层采用 CTP 或 TP Bypass 时，Read 事务支持 Request 切片，一个切片对应的响应为一个包。Read 事务不支持对 Read Response 切片，否则存在一个 Read Request 编号对应多个 Read Response 编号，出现异常。

- BTAH.TV\_EN：可设置为 0 或 1；
  - MAETAH.Address：读取 Target 内存地址；
  - MAETAH.Length：读取 Target 内存数据长度；
  - MAETAH.SO：可设置为 0 或 1，仅精简包格式下有效；
  - 当使用完整包格式时，包头中应额外包含如下信息：
    - (1) BTAH.ODR：无保序要求时，应设置为 3'b000，有保序要求时，BTAH.ORD 应设置为 3'b001 或 3'b010；
    - (2) BTAH.MT\_EN：应设置为 0；
    - (3) BTAH.INI\_RC\_Type：发起事务请求的 context 信息，仅当事务服务模式为 ROT 且申请到 SCID 后，才可设置为 2'b10，否则都应设置为 2'b00 或 2'b01；
    - (4) BTAH.INI\_RC\_ID：Target 返回 Read Response 时，需要携带该字段；
  - TAIDETAH.TAID：任务 ID 标识，可选字段，Target 返回 Read Response 时携带此域段；
  - OFSTETAH.Offset：表示请求包在事务层请求内的偏移量，可选字段，Target 返回 Read Response 时携带此域段，用于 Initiator 将 Read Response 数据写入本地内存。
2. Target 收到 Read 请求后，读取内存后返回 Read Response。若读取内存时出现异常，需要将异常信息携带在 Read Response 中返回给 Initiator。如果处理事务时出现资源不足，且传输层采用 RTP 时，可通过传输层反馈事务状态。

Read Response 包格式如下，详细信息见 7.2。



其中，

- ATAH.TAOpcode：应设置为 0x12；
- ATAH.INI\_TASSN：应与 Read 包中的 BTAH.INI\_TASSN 保持一致；
- ATAH.Status：可设置为 0 或 1；
- 当使用完整包格式时，包头中应额外包含如下信息：
  - (1) ATAH.INI\_RC\_Type：应与 Read 包中的 BTAH.INI\_RC\_Type 字段相同；
  - (2) ATAH.INI\_RC\_ID：应与 Read 包中的 BTAH.INI\_RC\_ID 字段相同；
  - (3) ATAH.RSPST：事务完成状态；
- TAIDETAH.TAID：任务 ID 标识，应与 Read 包中的 TAIDETAH.TAID 字段相同，方便 Initiator 定位哪个 Read Request 或切片得到响应；
- OFSTETAH.Offset：应与 Read 包中的 OFSTETAH.Offset 字段相同。

3. Initiator 收到 Read Response 后，事务操作结束。若返回的 Read Response 中 ATAH.Status 或 ATAH.RSPST 显示 Read 事务处理异常，进入异常处理流程，见 7.3.1.1。

### 7.4.2.3 Atomic 事务

#### 7.4.2.3.1 Atomic 概述

Initiator 使用 Atomic 事务对 Target 指定内存空间的数据做原子操作，包括读写、运算、交换等。Target 完成原子操作后通过 Atomic Response 返回给 Initiator，确认事务已经处理完成，返回 Initiator 需要的数据，并通告事务处理的状态，如正常处理、出现异常等。Atomic 事务执行应保障原子性，并且只执行一次。Atomic 事务仅包含一个包，支持 ROI、ROT、ROL 事务服务模式。

Atomic 事务包含多种事务子类型，包括：

- Atomic\_compare\_swap
- Atomic\_swap
- Atomic\_load
- Atomic\_store
- Atomic\_fetch\_add
- Atomic\_fetch\_sub
- Atomic\_fetch\_and
- Atomic\_fetch\_or
- Atomic\_fetch\_xor

Atomic 事务的处理流程图如下所示（以 Atomic\_load 为例，其它 Atomic 流程相同）：

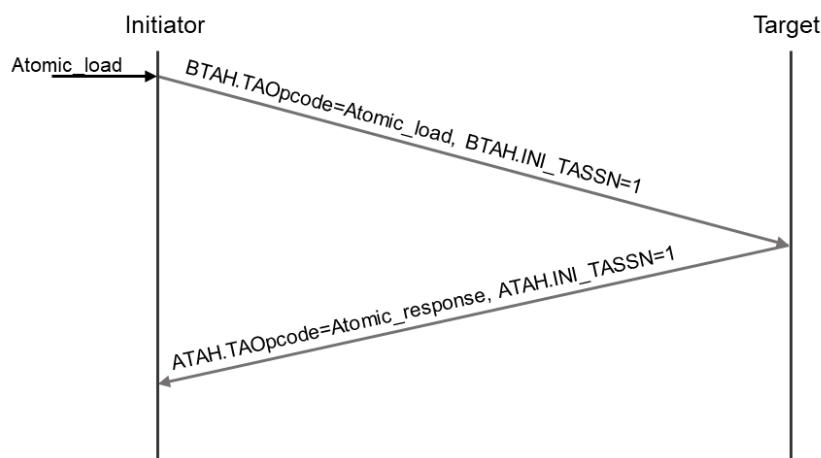
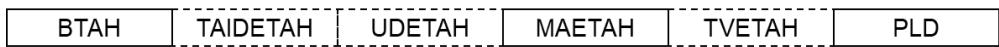


图 7-28 Atomic 事务处理流程

#### 7.4.2.3.2 Atomic Request

Initiator 发起 Atomic Request 到 Target。Atomic Request 包通过 Payload 域段承载 Atomic 事务所需的操作数。Payload 连续携带两个操作数，分别为第一操作数 Operand1 和第二操作数 Operand2。第一操作数与第二操作数长度相同。如果 Atomic Request 仅需使用第一操作数，则第二操作数为无效域段，但需保留在 Atomic Request 包中。

事务层请求包格式如下，详细信息见 7.2。



其中，

- BTAH.TAOpcode：根据不同的事务子类型设置，如下表：

表 7-16 Atomic 事务子类型

TAOpcode	Atomic 事务子类型	事务行为
0x7	Atomic_compare_swap	Initiator 将比较数据（第一操作数）和交换数据（第二操作数）发给 Target 侧，Target 将比较数据和指定的本地内存数据比较，如果相同则替换为交换数据，否则不替换，同时将 Target 交换前的数据返回给 Initiator。
0x8	Atomic_swap	Initiator 将交换数据（第一操作数）发送给 Target，Target 将交换数据与指定的本地内存数据交换，同时将 Target 交换前的数据返回给 Initiator。
0x9	Atomic_store	Initiator 将要运算数据（第一操作数）发送给 Target，Target 运算完成后，更新 Target 指定的本地内存，无需将计算前的值返回。支持的子事务类型如下（携带在 UDETAH 域段）： <ul style="list-style-type: none"> <li>• 0x0: ADD，将运算数据和指定的本地内存数据相加。</li> <li>• 0x1: CLR，将运算数据和指定的本地内存数据按位异或（XOR）。</li> <li>• 0x2: EOR，将运算数据按位取反后，再和指定的本地内存数据按位与（AND）。</li> <li>• 0x3: SET，将运算数据和指定的本地内存数据按位或（OR）。</li> <li>• 0x4: SMAX，将有符号的运算数与指定的本地内存数据相比，取较大者。</li> <li>• 0x5: SMIN，将有符号的运算数与指定的本地内存数据相比，取较小者。</li> <li>• 0x6: UMAX，将无符号的运算数与指定的本地内存数据相比，取较大者。</li> <li>• 0x7: UMIN，将无符号的运算数与指定的本地内存数据相比，取较小者。</li> </ul>

TAOpcode	Atomic 事务子类型	事务行为
0xA	Atomic_load	Initiator 将要运算数据（第一操作数）发送给 Target，Target 运算完成后，更新 Target 指定的本地内存，并将计算前的数据写回 Initiator 指定内存。支持的子事务类型和 Atomic_store 相同。
0xB	Atomic_fetch_add	Initiator 将要运算数据（第一操作数）发送给 Target，Target 将指定的本地内存的数据和运算数据相加，将计算结果写入 Target 指定的本地内存，同时将计算前的数据写回 Initiator 指定内存。
0xC	Atomic_fetch_sub	Initiator 将要运算数据（第一操作数）发送给 Target，Target 将指定的本地内存的数据和运算数据相减，将计算结果写入 Target 指定的本地内存，同时将计算前的数据写回 Initiator 指定内存。
0xD	Atomic_fetch_and	Initiator 将要运算数据（第一操作数）发送给 Target，Target 将指定的本地内存的数据和运算数据按位与，将计算结果写入 Target 指定的本地内存，同时将计算前的数据写回 Initiator 指定内存。
0xE	Atomic_fetch_or	Initiator 将要运算数据（第一操作数）发送给 Target，Target 将指定的本地内存的数据和运算数据按位或，将计算结果写入 Target 指定的本地内存，同时将计算前的数据写回 Initiator 指定内存。
0xF	Atomic_fetch_xor	Initiator 将要运算数据（第一操作数）发送给 Target，Target 将指定的本地内存的数据和运算数据按位异或，将计算结果写入 Target 指定的本地内存，同时将计算前的数据写回 Initiator 指定内存。

- BTAH.INI\_TASN：事务操作编号，Target 返回 Response 时需要携带，Initiator 根据该域段识别哪个事务请求得到响应；
- BTAH.UD\_Flag：可设置为 0 或 1，当事务子类型为 Atomic\_load 和 Atomic\_store 时，UDETAH[23:20]标识事务子类型；
- BTAH.TV\_EN：可设置为 0 或 1；
- MAETAH.Address：Target 内存地址，Atomic 事务时，要求 atomic 操作数是地址 aligned 的，例如，4 Bytes Atomic 事务的地址必须基于 4 Bytes 地址对齐；
- MAETAH.Length：Atomic 操作数长度，支持 1/2/4/8/16/32/64 Bytes；
- MAETAH.SO：可设置为 0 或 1，仅精简包格式下有效；
- 当使用完整头时，包头中应额外包含如下信息：
  - (1) BTAH.ODR：无保序要求时，应设置为 3'b000，有保序要求时，BTAH.ORD 应设置为 3'b001 或 3'b010；
  - (2) BTAH.MT\_EN：应设置为 0；

- (3) BTAH.INI\_RC\_Type: 发起事务请求的 context 类型, 仅当事务服务模式为 ROT 且申请到 SCID 后, 才可设置为 2'b10, 否则都应设置为 2'b00 或 2'b01;
- (4) BTAH.INI\_RC\_ID: 用于标记发起事务请求的上下文信息, 返回 TAACK 时, 应携带该字段;
- 可使用 TAIDETAH 携带 Task ID, 便于 Initiator 中查找具体事务, 如果 Request 中包含该扩展头的话, 需要在 Atomic Response 中返回该域段。
  - Payload 格式如下 (以操作数长度为 8 Bytes 的 Atomic\_compare\_swap Request 为例):

Byte0								Byte1								Byte2								Byte3							
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
Operand1[7:0]								Operand1[15:8]								Operand1[23:16]								Operand1[31:24]							
Operand1[39:32]								Operand1[47:40]								Operand1[55:48]								Operand1[63:56]							
Operand2[7:0]								Operand2[15:8]								Operand2[23:16]								Operand2[31:24]							
Operand2[39:32]								Operand2[47:40]								Operand2[55:48]								Operand2[63:56]							

#### 7.4.2.3.3 Atomic Response

Target 收到 Atomic Request 后, 处理完成后需要返回 Atomic Response, 若出现事务处理异常, 需要将异常信息携带在 Atomic Response 包头中。如果反馈 Response 时出现资源不足, 且传输层采用 RTP 时, 可通过传输层反馈事务状态。

Atomic Response 包格式如下, 详细信息见 7.2。



其中,

- ATAH.TAOpcode: 应设置为 0x13;
- ATAH.INI\_TASSN: 应与 Atomic Request 包中的 BTAH.INI\_TASSN 保持一致;
- ATAH.Status: 携带 Target 鉴权结果, 仅在精简包格式中有效;
- 当使用完整头时, 包头中应额外包含如下信息:
  - ATAH.INI\_RC\_ID: 应与 Atomic Request 包中的 BTAH.INI\_RC\_ID 字段相同;
  - ATAH.RSPST: 事务层完成状态;
- TAIDETAH: 应与 Request 的 TAIDETAH 扩展头内容保持一致。

当 Target 收到的 Atomic Request 的事务类型不为 Atomic\_store 时, Atomic Response 需要额外携带 Payload。Payload 携带 Atomic 事务对应的原始数据, 数据长度与 Atomic Request 的操作数长度一致, 长度支持 1/2/4/8/16/32/64 Bytes。以 8 Bytes 的原始数据为例, 格式如下:

Byte0								Byte1								Byte2								Byte3							
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
Origin_data[63:56]								Origin_data[55:48]								Origin_data[47:40]								Origin_data[39:32]							
Origin_data[31:24]								Origin_data[23:16]								Origin_data[15:8]								Origin_data[7:0]							

Initiator 收到 Atomic Response 后，事务操作结束。若返回的 Atomic Response 中 ATAH.Status 或 ATAH.RSPST 显示 Atomic 事务处理异常，进入异常处理流程，见 7.3.1.1。

## 7.4.3 消息事务

### 7.4.3.1 Send

Initiator 使用 Send 事务发送消息到 Target, Send 事务由 Initiator SQ 发起，并需要消耗 Target RQE，RQE 中包含事务操作信息。事务支持 ROI、ROT、ROL、UNO 事务服务模式。

Send 事务长度应满足如下约束：

1. 当传输层采用 CTP 或 TP Bypass 时，Send 事务只包含一个包；当传输层采用 RTP 时，Send 事务长度无该约束，但仅支持一个事务切片，一个事务切片经过相同 TP Channel 传输。
2. 当采用 UNO 模式时，Send 事务只包含一个包。

Send 事务的处理流程图如下所示：

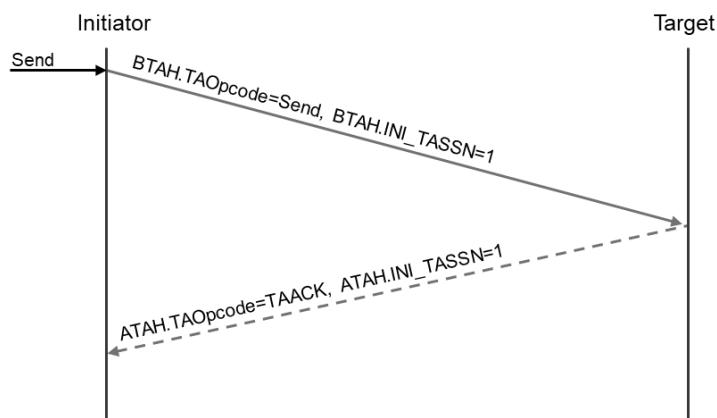


图 7-29 Send 事务处理流程

1. Initiator 发起 Send 事务到 Target。事务层请求包格式如下，详细信息见 7.2。



其中：

- BTAH.TAOpcode：应设置为 0x0
- BTAH.INI\_TASSN：事务操作编号，Target 返回 TAACK 时，需要携带，Initiator 根据该域段识别哪个事务请求得到响应；  
注：当传输层采用 RTP 时，事务仅包含一个切片，否则 Target 在等待每个切片全部数据收齐时，可能等待时间过长导致耗费资源过多；当传输层采用 CTP 或 TP Bypass 时，Send 事务仅支持一个包。
- BTAH.TV\_EN：可设置为 0 或 1；
- BTAH.UD\_Flag：可设置为 0 或 1；
- 当使用完整包格式时，包头中应额外包含如下信息：
  - (1) BTAH.ODR：无执行序要求时，BTAH.ODR[1:0]应设置为 2'b00，有保序要求时，BTAH.ORD[1:0]应设置为 2'b01 或 2'b10；无完成序要求时，BTAH.ODR[2]应设置为 0，有完成序要求时，BTAH.ODR[2]应设置为 1；
  - (2) BTAH.INI\_RC\_Type：仅当事务服务模式为 ROT 且申请到 SCID 后，才可设置为 2'b10，否则都应设置为 2'b00 或 2'b01；
  - (3) BTAH.INI\_RC\_ID：Requester Context 标识，Target 返回 TAACK 时，需要携带该字段；
  - (4) BTAH.MT\_EN：应设置为 1b'1；
- MTETAH.TGT\_TC\_Type：可设置为 TC，或 TC Group；
- MTETAH.TGT\_TC\_ID：Target Context 标识；
- OFSTETAH.Offset：表示 Send 包在事务层请求内的偏移量，以 1KB 为粒度，首个包 offset 为 0，第二个及后续包 offset 按 1KB 粒度依次递增。使用 CTP 时，该域段为 Reserved。

2. Target 收到 Send 包后，进行消息处理，处理完成后 Target 返回 TAACK，若处理过程中出现异常，需要将异常信息携带在 TAACK 中返回给 Initiator。当事务服务模式为 ROI、ROT、ROL 时，TAACK 特殊处理可见 7.3.1.1；当事务服务模式为 UNO，不返回 TAACK。  
TAACK 包格式如下，详细信息见 7.2 节。

### ATAH

- (1) ATAH.TAOpcode：应设置为 0x11；
- (2) ATAH.INI\_TASSN：应与 Send 包中的 BTAH.INI\_TASSN 保持一致；
- (3) ATAH.INI\_RC\_Type：应与 Send 包中的 BTAH.INI\_RC\_Type 保持一致；
- (4) ATAH.INI\_RC\_ID：应与 Send 包的 BTAH.INI\_RC\_ID 段相同；
- (5) ATAH.RSPST：事务层完成状态。
3. Initiator 收到 TAACK 后，事务操作结束，若返回的 TAACK 中 ATAH.Status 或 ATAH.RSPST 显示事务处理异常，进入异常处理流程，见 7.3.1.1。

### 7.4.3.2 Send\_with\_immediate

Initiator 使用 Send\_with\_immediate 事务发送消息到 Target，并且最后一个包中除携带 Send 数据外，应携带 immediate 数据。Immediate 数据只有事务中所有的 Send 包被处理完才能被处理，Immediate 数据作为通知信息（CQE）直接通知 Target 处理。

Send\_with\_immediate 事务可使用 ROI、ROT、ROL、UNO 事务服务模式。

和 Send 事务相同，当传输层采用 CTP 时，Send\_with\_immediate 事务仅支持一个包；当传输层采用 RTP 时，不支持事务切片，即所有包采用相同 TASSN，经过相同的 TP Channel 传输，通过 TP Channel 可实现保序。

Send\_with\_immediate 的处理流程图如下：

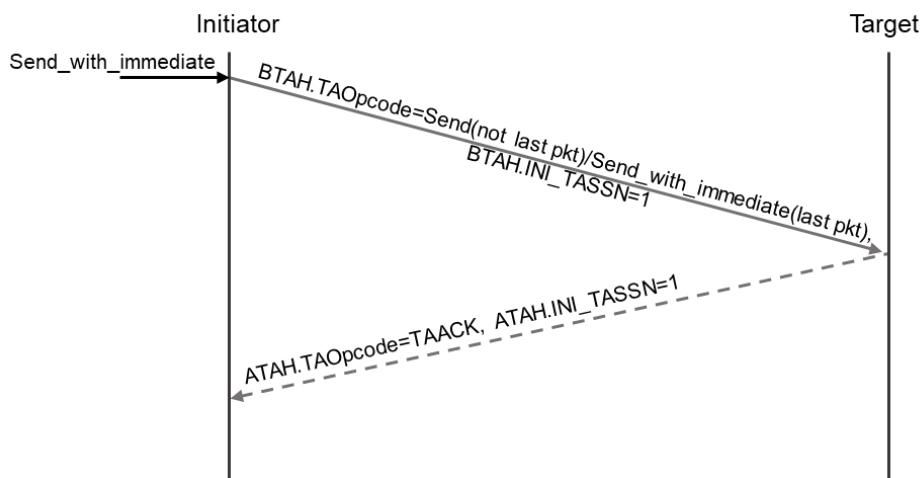


图 7-30 Send\_with\_immediate 事务处理流程

1. Initiator 发起 Send\_with\_immediate 事务。若事务包含多个包，则前面的包为 Send 包，最后一个包除携带 Send 数据外，应携带 Immediate 数据；若事务仅包含一个包，则包中应同时携带 Send 数据以及 Immediate 数据。

Send 包格式以及域段设置同 Send 事务，见 7.4.3.1。

最后一个包中应携带 Immediate 数据。包格式如下，详细信息见 7.2。



其中，

- (1) BTAH.TAOpcde：应设置为 0x1；
- (2) BTAH.UD\_Flag：可设置为 0 或 1；
- (3) 应包括 IMMETAH 包头，包头中应包含 Immediate 数据(8B)、TokenValue 以及 Msg\_length；
- (4) BTAH.MT\_EN：应设置为 1；
- (5) MTETAH.TGT\_TC\_ID：应设置为处理 Immediate Data 的 Target 的 TCID；

2. Target 收到 Send 数据以及 Immediate 数据，根据场景要求决定是否返回 TAACK（是否返回同 Send 事务相同）。TAACK 包格式同 Send 操作，见 7.4.3.1。
3. Initiator 收到 TAACK 后，事务操作结束，若返回的 TAACK 中 ATAH.Status 或 ATAH.RSPST 显示事务处理异常，进入异常处理流程，见 7.3.1.1。

#### 7.4.3.3 Write\_with\_immediate

Initiator 使用 Write\_with\_immediate 事务将本地数据写入 Target 的指定内存空间，并且最后一个包中除可携带 Write 数据外，应携带 Immediate 数据，同 Send 事务不同，仅携带 Immediate 数据的包处理需要消耗 Target RQE。Immediate 数据只有事务中所有包被处理完才能被处理，Immediate 数据不需要写入指定内存中，而是作为通知信息（CQE）直接通知 Target 处理。

Write\_with\_immediate 事务可使用 ROI、ROT、ROL 事务服务模式。

Write\_with\_immediate 在不同事务服务模式下的处理流程与 Write\_with\_notify（见 7.4.2.1.2）相同。差异点如下：Write\_with\_immediate 的 Immediate 包的包格式同 Notify 包的包格式不同，Immediate 包的包格式如下，详细信息见 7.2。

BTAH	UDETAH	MTETAH	MAETAH	IMMETAH	TVETAH	PLD
------	--------	--------	--------	---------	--------	-----

其中，

- BTAH.TAOpcode：应设置为 0x4；
- BTAH.ODR：无完成序要求时，BTAH.ODR[2]应设置为 0，有完成序要求时，BTAH.ODR[2]应设置为 1；
- BTAH.UD\_Flag：可设置为 0 或 1；
- BTAH.MT\_EN：应设置为 1；
- 应包括 IMMETAH 包头，包头中应包含 Immediate 数据，以及地址段对应的访问凭据等。

#### 7.4.4 维护事务

##### 7.4.4.1 Prefetch\_tgt

Initiator 通过 Prefetch\_tgt 事务对 Target 数据进行预读，以减少后续 Initiator 对 Target 的读操作延时。Prefetch\_tgt 事务仅包含一个包，Prefetch\_tgt 事务可使用 UNO 模式。

Prefetch\_tgt 事务的处理流程图如下所示：

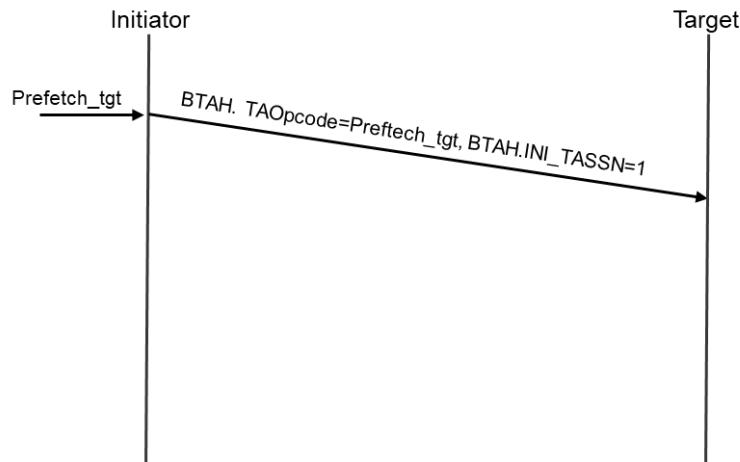


图 7-31 Prefetch\_tgt 事务处理流程

1. Initiator 发起 Prefetch\_tgt 事务到 Target。事务层请求包格式如下，详细信息见 7.2。



其中：

- (1) BTAH.TAOOpcode：应设置为 0x15；
  - (2) BTAH.UD\_Flag：可设置为 0 或 1；
  - (3) MAETAH.Address：Target 要 Prefetch 的内存地址。
2. Target 收到 Prefetch\_tgt 事务后无需返回 TAACK，可选是否对数据进行预取。
  3. Initiator 不会发起 Prefetch\_tgt 重试，如果通信过程中检测到异常，直接静默丢弃，不需要上报任何异常。

## 7.4.5 管理事务

### 7.4.5.1 Management

Management 事务用于承载管理命令（见 10.4.3），一般使用 UNO 模式，无可靠性以及保序的要求。

Management 事务的交互流程如下：

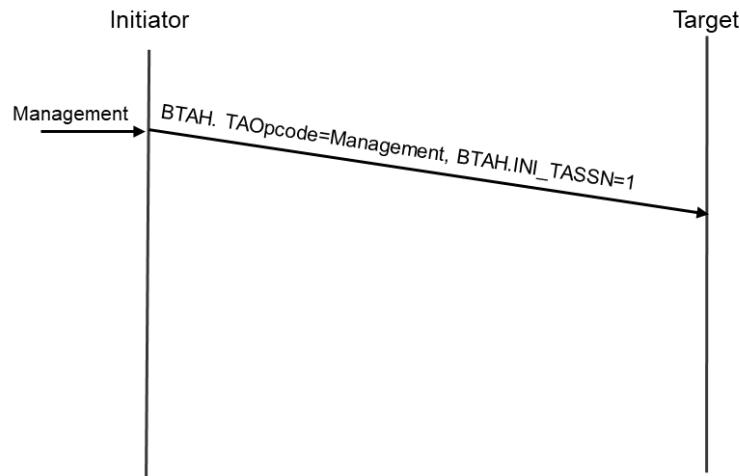
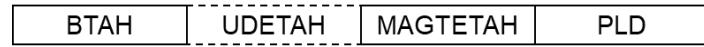


图 7-32 Management 事务处理流程

Management 事务包格式如下，详细信息见 7.2。



其中：

- BTAH.TAOpcode：应设置为 0x10；
- BTAH.TV\_EN：应设置为 0；
- BTAH.INI\_TASSN：代表请求的 Message Sequence Number ( MSN )；
- BTAH.INI\_RC\_ID：设定为 message queue index ( 上限根据实现决定 )，当上层管理软件返回响应时，需要携带该字段，用于索引正确的事务发送方。
- MGMTEATAH：具体见 7.2.12。

# 8 功能层



## 8.1 概述

功能层位于事务层之上，通过调用事务层能力实现各种功能。

UB 功能层包含两类内容：编程模型以及基于编程模型构建的高级功能。开发者可通过编程模型直接调用事务层能力，完成各种事务操作；UB 功能层也提供了一些高级功能，简化用户开发，提升使用效率。UB 支持用户自定义编程模型以及高级功能，用户也可选使用 UB 功能层已定义内容。

针对编程模型，UB 功能层支持 Load/Store 同步访问以及 URMA 异步访问两种编程模型。使用两种模型均可调用事务层事务操作，包括内存事务、消息事务、维护事务、管理事务。对于 Load/Store 同步访问，UB Controller 配合片上总线将 Load/Store 指令转换为事务操作（如 Read、Write、Atomic 等）；对于 URMA 异步访问，应用可调用 Jetty 提供的接口，完成通信对关系建立、事务操作提交与查询响应等功能。URMA 编程模型需要绑定事务队列或其它载体用于承载 Jetty 事务请求。

UB 功能层也提供了三种高级功能，包括统一远程过程调用（URPC）、多 Entity 协同以及 Entity 管理。URPC 基于 Load/Store 或 URMA 编程模型提供的编程能力，支持 UBPU 之间通过统一的函数接口平等互访；多 Entity 协同通过灵活组合事务层机制实现更高效的通信任务，如融合操作、集合通信操作以及全局维护等；Entity 管理从单 UBPU 角度调用事务层能力完成 Entity 管理配置，包括本地 Entity 发现、池化 Entity 注册、配置管理、中断与消息通知、通信与远端内存注册控制、虚拟化等管理功能。

## 8.2 基本概念

### 8.2.1 内存段

#### 8.2.1.1 内存段创建与删除

内存段是一段连续地址的内存。内存段提供方应用通过调用 OS 的内存分配接口，创建一个内存段对象，生成访问此内存段所需要的 EID、TokenID、UBA，以及内存段对象的大小。在创建内存段对象时，UB 内存管理单元（UMMU）为内存段分配物理内存，但内存管理系统也可能采用延迟分配策略，即并不在创建内存段时分配物理内存，而是在内存段真正被访问到时才分配。多个内存段可共享同一个 TokenID，也可以为一个内存段对象生成不同的 TokenID。

内存段访问时，内存访问发起方（User）为 Initiator，向事务层提交事务请求，内存段提供方（Home）为 Target，接收并处理 Initiator 发起的内存事务。

Home 侧在内存段创建过程中，UMMU 为该内存段配置地址转换表 MATT 和权限校验表 MAPT 的过程具体可见内存管理章节（见 9.4 节）。

Home 侧删除内存段时，要保证 UB 网络中残留的与该内存段相关的事务层包不会导致数据不一致。因此，在清理 UMMU 内表项之前，需要注意与该内存段相关的所有内存访问事务都已经被处理完成。

### 8.2.1.2 内存段使用

Initiator 访问 Target 内存段前，需要先从 Target 获取内存段信息，获取方式见 8.2.5 节

Initiator 获取到内存段信息后，有两种使用方式：

1. Initiator 将内存段的地址信息映射到本地进程的 VA 空间中，得到 MVA( Mapped Virtual Address )。  
Initiator 可使用同步或异步访存方式访问该内存空间，见 7.4 节；
2. Initiator 只申请使用，不做映射。Initiator 可使用异步访问模型访问该内存空间，见 7.4 节。

Initiator 访问内存段时，应向事务层传递内存段信息（含 EID、TokenID、UBA 等），并封装在事务层包中，便于 Target 内存地址访问。如果 Target 对于内存段开启安全保护，Initiator 应附加 TokenValue 信息作为内存段信息发送给 Target 用于在 Target 做安全校验。访问安全机制见 8.2.4 节。

### 8.2.1.3 内存段访问方式

支持通过两种方式访问 Target 内存段：

1. 内存首地址以及长度

当 Initiator 要访问内存段的某段连续地址时，可以指定访问的首地址以及访问长度信息。是否需要地址对齐，见具体事务要求（见 7.4 节），如 4 Bytes Atomic 事务中，要求 4 Bytes 地址对齐。

2. ByteEnable

当 Initiator 要访问内存段的某些字节时，如仅更新某个字节时，为提高访存效率，事务层支持通过 ByteEnable 方式指定要更新的具体字节，只有标记为有效的字节才更新，无效字节不更新，如 Write\_with\_be 事务。

## 8.2.2 Jetty

### 8.2.2.1 Jetty 概述

Jetty 是 URMA 基本通信单元，位于事务层之上，开发者可通过 Jetty 发起事务操作。使用 Jetty 调用事务操作时，需要调用接口创建，并导入 Target Jetty 和内存段。通过 Jetty 可以发起内存事务以及消息事务等。同时，应绑定事务队列用于承载 Jetty 事务，详细使用方式见 8.2.3。

根据 Jetty 的组成不同，用于事务发送/接收的 Jetty 可分为三种类型：

- **Type 1: 标准 Jetty**

Jetty 既可以用于发送事务请求，也可以用于接收事务响应，由唯一的 Jetty ID 标识。使用标准 Jetty 时，需要将 Jetty 绑定 SQ 以及 RQ，用于完成事务操作。

- **Type 2: 单边 Jetty**

单边 Jetty 是标准 Jetty 的一种简化版，用于仅需要单向访问的场景中，与标准 Jetty 相比，可以只绑定到 SQ 或 RQ 中，以减少资源的占用。单边 Jetty 包括 JFS（Jetty For Sending）、JFR（Jetty For Receiving）。在使用单边 Jetty 访问时，Initiator 仅发送事务，Target 仅接收事务或处理事务请求。JFS 仅绑定 SQ，JFR 仅绑定 RQ。注：当 Initiator 使用 JFS 访问 Target 内存段时，无需 Target 创建 JFR。

- **Type 3: Jetty Group**

Jetty Group 是一种特殊的 Jetty 类型，仅存在于 Target。Jetty Group 中包含多个 Jetty，并分配唯一的 Jetty Group ID。Jetty Group 中包含的 Jetty 可选择使用标准 Jetty 或者 JFR，Jetty Group 需要绑定多个 RQ，即 RQ Group。Initiator 发送事务请求时，不需要指定单独的 Jetty，仅需要指定 Jetty Group 即可。Target 收到事务请求后，会根据策略将事务请求分发至不同的 Jetty。Jetty Group 的分发处理不需要 CPU 参与。

使用 Jetty Group 有如下两个优点：

- (1) 从 Jetty 收到的事务请求分发到不同的线程去处理时，可以免 CPU 参与请求分发，避免独立 CPU 完成线程分发时的资源浪费；
- (2) 分发时选择和线程 NUMA 亲和的 Jetty 处理，避免跨 NUMA 访问。

当 Target 多个 Jetty 满足要求时，Target 支持根据多种策略选择 Target Jetty（具体使用的策略在创建 Jetty Group 时指定），即选择 Target Jetty 或 JFR，包括：

- (1) Initiator 指定 Hint 并携带在包头中，Target 根据 Hint 域段进行哈希；
- (2) 按照 Round Robin 方式选择 Jetty；
- (3) 按照 RQ 深度动态做负载均衡，选择空闲 RQ 对应的 Jetty 处理。

Jetty Group 模型如下所示。

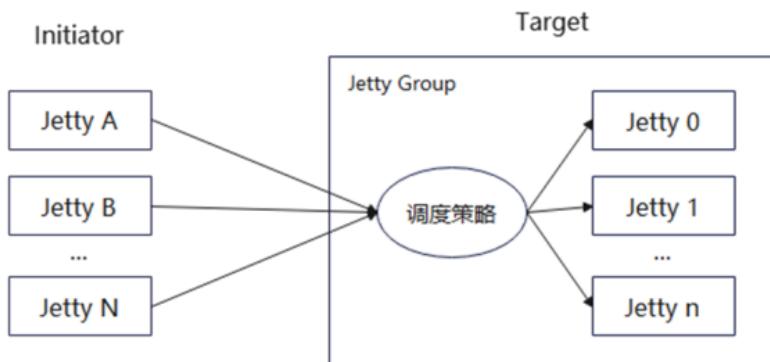


图 8-1 Jetty Group 模型

除 Jetty 外，URMA 异步访问编程模型还提供 Jetty For Completion ( JFC )、Jetty For Completion Event ( JFCE ) 以及 Jetty For Asynchronous Event ( JFAE ) 机制，方便用户获取事务执行结果或其他异常类事件。

JFC 用于接收某一 Jetty 的事务完成通知，需要绑定一个唯一的 CQ。当事务操作完成后，事务层会按需向 JFC 中提交一个 CQE，用于通知事务完成并携带事务执行结果或状态信息。UB 支持多个 Jetty 共享一个 JFC，即 CQ 中可存储多个 Jetty 的完成通知。用户可通过轮询 JFC 的方式获取事务操作的完成状态。

JFCE 用于通过中断的方式通知用户事务完成状态，与某一个 JFC 绑定，同时 JFCE 需要绑定一个唯一的 EQ，用于承载 Event 事件。当采用中断方式向上通知时，可以在事务请求中下发 FCE 标记，事务完成后立即产生一个 Event 事件，也可以通过定时器或完成通知数量超过阈值等机制产生 Event 事件。

JFAE 用于接收异步事件上报，用户可通过 JFAE 获取各种异常事件，包括 Jetty 异常或者驱动硬件异常等。JFAE 需要绑定一个 EQ。

### 8.2.2.2 Jetty 通信关系

根据 Initiator 和 Target 通信时的通信关系，Jetty 可以分为多对多通信以及一对一通信两种类型。无论采用哪种类型，Initiator 和 Target 均需要交互 Jetty 类型以及 Jetty ID 等信息。

- **类型 1：多对多通信模型**

Initiator 的 Jetty 与 Target Jetty 无一对一绑定关系，Initiator Jetty 可以和任意的 Target Jetty 通信，Target Jetty 也可以接收来自任意 Initiator Jetty 的事务请求。多对多通信模型也可称为 M2N 通信模型。使用多对多通信模型，可大幅减少 Jetty 数量，避免 Jetty 资源浪费。由于 Initiator Jetty 可与任意 Target Jetty 通信，因此当 Initiator 发起事务请求时，需要指定 Target Jetty。当 Target Jetty 返回响应时，需要将 Initiator Jetty 信息携带在事务响应中。

标准 Jetty 以及单边 Jetty 均可使用多对多通信模型，模型如下：

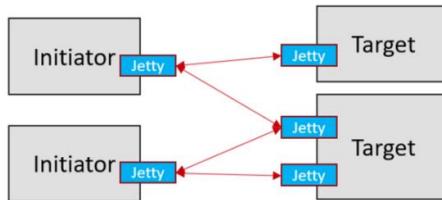


图 8-2 Jetty 多对多通信模型

当 Initiator 发起异步访问时，过程可参考如下。

- (1) Initiator 获取 Target 的 Jetty 信息并申请使用，包含 Jetty 类型以及 Jetty ID。Jetty 信息交互流程见 8.2.5 节。
- (2) Initiator 事务请求发送支持任意 Target，事务请求中应指定 Target Jetty 信息；
- (3) Target 支持接收来自任意 Initiator 的事务，事务响应中应携带 Initiator Jetty 信息，方便 Initiator 处理。

- **类型 2：一对通信模型**

Initiator 的 Jetty 与 Target Jetty 有一对一绑定关系，Initiator Jetty 仅能和绑定的 Target Jetty 通信，Target Jetty 也仅能接收绑定的 Initiator Jetty 的事务请求。

仅标准 Jetty 可以使用一对通信模型，模型如下：

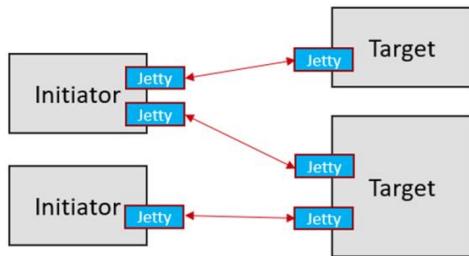


图 8-3 Jetty 一对通信模型

当 Initiator 发起异步访问时，访问过程同多对多通信模型相同，但限制 Initiator 和 Target Jetty 仅能申请使用唯一的 Jetty。

### 8.2.2.3 Jetty 状态机

Jetty 状态机包含 Reset、Ready、Suspend 和 Error 四种状态。应用可以通过调用 Jetty 销毁接口从任何状态退出 Jetty 状态机。Jetty 状态机下图所示：

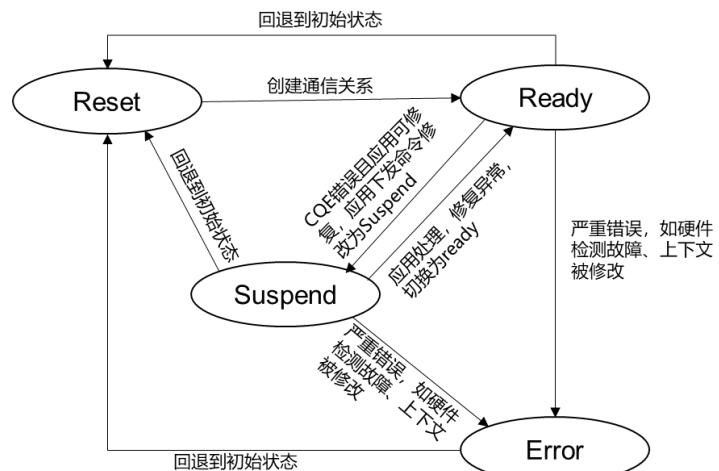


图 8-4 Jetty 状态机

#### 1. Reset 状态

新创建的 Jetty 应置于 Reset 状态。Reset 状态下会为 Jetty 分配相应的资源空间。当 Jetty 跳转到 Reset 状态时，Jetty 的属性将被初始化。

Jetty 可以从任何状态跳转至 Reset 状态。应用通过调用 Jetty 修改接口从 Reset 状态跳转至 Ready 状态。

处于 Reset 状态下的 Jetty 接收到 Target 的事务层包时，该事务层包将被静默丢弃；处于 Reset 状态下的 Jetty 接收到 Work Request 时，Jetty 会返回错误信息。

## 2. Ready 状态

在转换到 Ready 状态之前，必须完成 Jetty 的建链协商。

处于 Ready 状态下的 Jetty 接收到 Target 的事务层请求时，Jetty 应正常执行该事务层请求；处于 Ready 状态下的 Jetty 接收到 Work Request 时，Jetty 应正常执行该 Work Request。

当发送过程中出现可恢复的故障，如传输通道出现故障，Jetty 将由 Ready 状态跳转至 Suspend 状态。当发送过程中出现严重错误，如 Jetty 上下文遭到篡改，Jetty 将由 Ready 状态跳转至 Error 状态。

## 3. Suspend 状态

Jetty 支持配置 Exception Mode。Exception Mode 分为 Exception continue 和 Exception suspend 两种。Exception continue 表示 Jetty 出现错误后，不需要用户任何处理，Jetty 还可继续工作；Exception suspend 表示 Jetty 出现错误后，需要被用户处理后才可恢复继续工作。

如果 Exception mode 配置为 Exception continue，当检测到 Send Queue Element ( SQE ) 异常或接收到错误应答时 Jetty 状态将不受影响，并基于 SQE 粒度上报异常 CQE，后续 SQE 可继续正常执行。该模式下 Jetty 状态机不存在 Suspend 状态，即 Jetty 不会进入 Suspend 状态，出现上述错误时 Jetty 仍处于 Ready 状态。

如果 Exception mode 配置为 Exception suspend，当检测到 SQE 异常或接收到错误应答时应上报相应的异常 CQE 或异步错误，Jetty 将由 Ready 状态跳转至 Suspend 状态。处于 Suspend 状态下的 Jetty 会暂停处理新的 Work Request，停止调度新的 SQE 进入传输通道，并等待已经分发的 SQE 的执行结果，上报完 CQE 后对 SQ 进行排空处理。处于 Suspend 状态下的 Jetty 仍可继续接收 Target 的事务层包。如果状态跳转时 SQ 未完全排空，则 Jetty 会上报错误。

- 应用处理故障完成后，会将 Jetty 恢复。此时 Jetty 将由 Suspend 状态跳转至 Ready 状态，继续正常的收发工作。
- 当 Jetty 无法恢复或应用主动置错时，Jetty 将由 Suspend 状态跳转至 Error 状态，此时 Jetty 将无法工作。

## 4. Error 状态

处于 Error 状态的 Jetty 将停止 SQE 的调度分发和事务层包的接收。处于 Error 状态下的 Jetty 接收到 Target 的事务层包时，该事务层包将被静默丢弃；处于 Error 状态下的 Jetty 接收到 Work Request 时，Jetty 会返回错误信息。

应用会通过调用 Jetty 修改接口从 Error 状态跳转至 Reset 状态。在状态跳转前，所有 SQE 需完成错误 CQE 的上报。

### 8.2.3 事务队列

事务队列是承载异步访问编程模型的载体之一，是软硬件操作媒介。当事务队列承载 URMA 异步访问编程模型时，需要将 Jetty、JFC 以及 JFCE 与事务队列绑定，用户通过 Jetty 即可使用事务队列。事务队列采用先入先出（First In First Out, FIFO）工作模式，因此处于相同队列的事务可以做事务序要求，即相同队列的事务可以有保序要求，不同队列的事务无保序要求。

事务队列分为发送队列（Send Queue, SQ），接收队列（Receive Queue, RQ），完成队列（Complete Queue, CQ），事件队列（Event Queue, EQ）四种。四种队列是否使用根据事务请求决定，如当执行内存事务时，可直接指定内存，而不需要绑定接收队列 RQ。

1. 发送队列 SQ 用于保存用户提交的事务请求，可包含一个或多个事务请求，即 SQE，每个 SQ 对应单独的 RCID（Requester Context ID）；
2. 接收队列 RQ 用于保存待处理的事务请求，可包含一个或多个 RQE，每个 RQE 中保存事务接收后处理的上下文信息，每个 RQ 对应单独的 TCID（Target Context ID）。多个 RQ 可以绑定成一个 RQ Group，每个 RQ Group 对应单独的 TC Group ID；
3. 完成队列 CQ 用于保存已经处理完成的事务，可包含一个或多个 CQE，每个 CQE 中包含事务执行的状态，和 SQ 以及 RQ 搭配使用，用户通过轮询 CQ 获得事务的执行状态；
4. 事件队列 EQ 用于保存事件通知，可以中断方式通知用户处理。

当事务队列调用事务层能力时，需要按需将 RCID 以及 TCID 下发给事务层，供事务层包封装使用，该信息用于标识事务发起单元，或者用于将事务传递到对应的 Jetty 上处理。

### 8.2.4 访问安全

Initiator 在发送事务请求前应获取 Target Jetty 或内存段的访问凭据，可采用三种方式查找：

1. 从 Initiator SQE 中获取 Target Jetty 的 TCID 和 TokenValue。
2. 从 Initiator SQE 中获取内存段的 TokenID 和 TokenValue。
3. 通过 UB Decoder 的转换结果获取到 TokenID 和 TokenValue。

依据访问凭据进行权限校验的具体机制参见安全章节。

### 8.2.5 通信管理

Initiator 和 Target 需要交换信息，包括内存段以及 Jetty 的信息，可采用如下方式完成信息交换，也可扩展其它交换方式。不管哪种方式，信息交互前应已建立物理节点之间的传输通道。

1. 通过 UBFM 完成信息交换。
2. Initiator 使用公知 Jetty 的通信操作与 Target 交互，获取 Target 的内存段信息或 Jetty 信息，流程如下图所示：

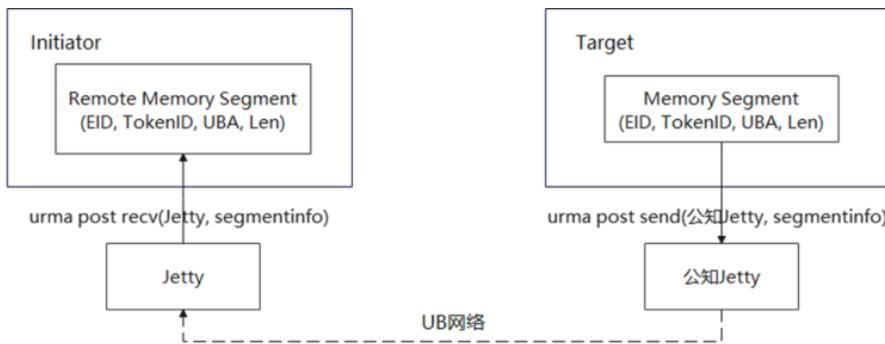


图 8-5 公知 Jetty 通信交互流程

公知 Jetty 是指 Initiator 通过约定或配置的方式获取到的 Target Jetty。Target 在创建公知 Jetty 时，可指定公知 Jetty 号，Initiator 使用本地 Jetty 与 Target 的公知 Jetty 交互。公知 Jetty 只能被某个进程独占，如果有多个 Target 进程，则创建公知 Jetty 时应指定不同的 Jetty 号。预留的公知 Jetty 编号如下表所示。

表 8-1 公知 Jetty 编号

Reserved Jetty ID	Usage description
0	交换传输层信息
1	交换事务层信息
2	Socket over UB
3-31	Reserved
32-1023	User defined

3. Initiator 使用 TCP/IP 连接来交换通信信息，获取 Target 的内存段信息或 Jetty 信息，流程如下图所示：

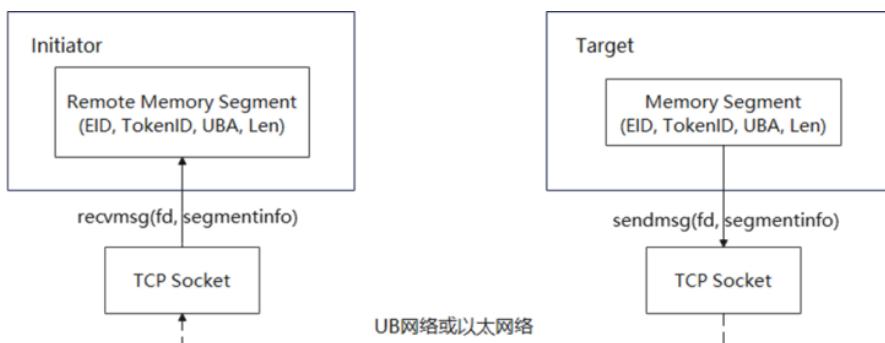


图 8-6 TCP/IP 通信交互流程

## 8.2.6 内存借用与共享

开发者可基于 UB 事务层以及功能层编程模型构建内存池，可提升单节点可用内存容量以及可用内存带宽。基于 UB 平等架构，构成内存池的每个 UBPU 节点既可以作为内存借用方，访问其它节点提供的内

存，也可以作为内存借出方，为其它节点提供内存。并且内存访问关系是动态的，即根据业务需要灵活申请、释放，以提升内存使用效率。

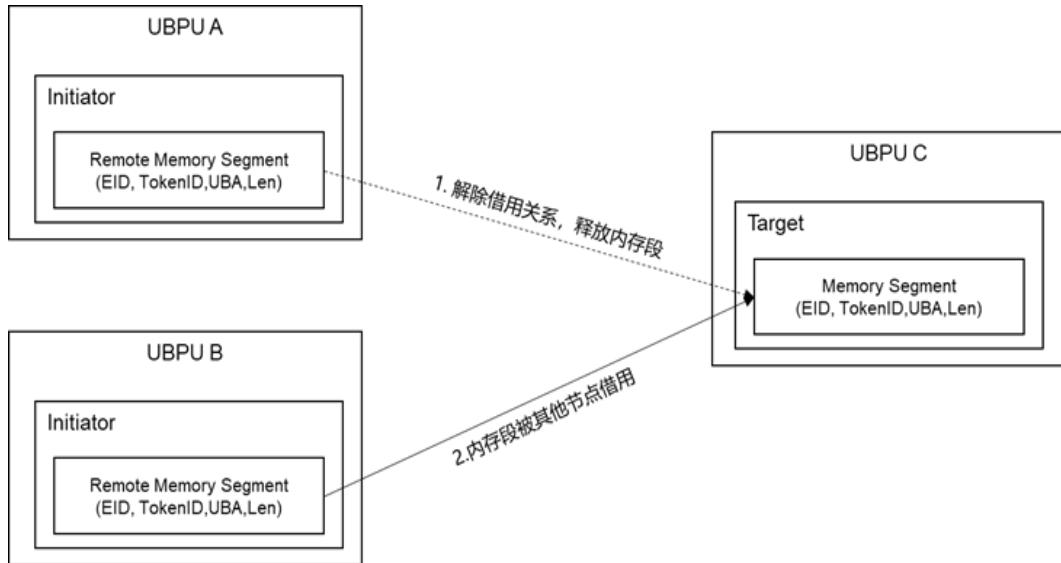


图 8-7 内存动态申请释放

UB 支持内存借用以及内存共享两种方式构建内存池：

1. 内存借用方式，内存借用方可借用其它节点内存作为本地内存使用，且内存借出方某一内存段仅可被唯一的内存借用方访问；
2. 内存共享方式，内存提供方某一内存段可以被多个内存使用方共享访问。

Initiator 使用 Target 的内存时，可采用两种使用方式：

1. Cacheable 方式，即 Initiator 访问 Target 内存时，Cache 一致性和本地内存能力相同。
2. Non-Cacheable 方式，即 Initiator 访问 Target 内存时，避免 Cache 一致性管理问题，简化访问通路。

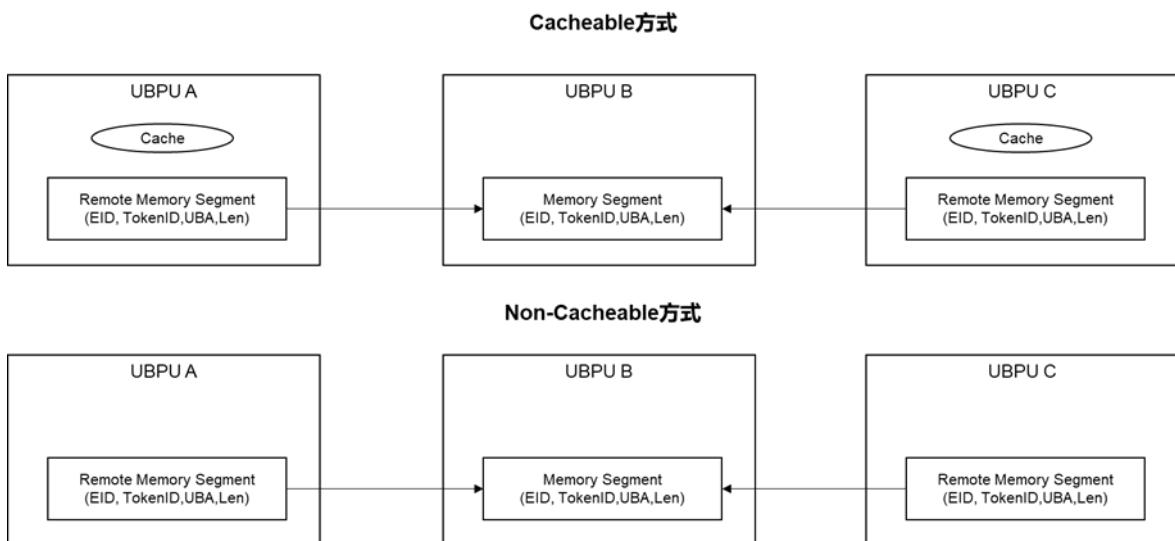


图 8-8 内存池化使用方式

对于具有局部性的程序，UB 支持以 Cacheable 方式访问远端内存，即把数据缓存在本地 Cache，有利于保持程序性能。对于以通信为主的程序，访问远端内存主要是交换数据，以 Non-Cacheable 方式访问远端内存可以带来更好的性能。

通常情况下，内存借用场景下使用 Cacheable 方式，内存段仅由唯一的 Initiator 访问，不存在 Cache 不一致问题。内存共享场景可选用 Cacheable 或 Non-Cacheable 方式，但对于内存共享方式下的 Cacheable 方式，由于内存段被多个节点使用，需要保证多节点的 Cache 一致性。UB 提供 Ownership 机制，提供对共享内存的读写控制访问权限，保证跨节点数据一致性。Ownership 管理机制可以基于硬件机制也可以采用软件实现，UB 协议对具体机制协议不做规定。

Cacheable 模式下，多节点共享内存场景的软硬件协同缓存一致性的一种参考实现如下：

软件定义 Ownership 概念，用来管理任意粒度内存段的持有状态。每个节点对于一个内存段均有独立的本地 Ownership 状态，分为三类：

- Invalid: 本节点不能对内存进行读写操作；
- Write: 本节点可以对内存进行读操作或者写操作；
- Read: 本节点可以对内存进行读操作；

注意，同一时刻当有一个节点为 Write Ownership 状态时，其它所有节点必须为 Invalid Ownership 状态。

用户需要根据内存读写的诉求主动转换 Ownership 状态，保证软件对于内存的读写符合上述 Ownership 状态的定义。Ownership 状态转移操作说明：

- Write->Invalid: Clean&Invalidate，即确保修改的数据写回内存，并且对应的 Cache 数据已经失效。
- Write->Read: Clean，即确保修改的数据写回内存，同时保留对应的 Cache 数据，用于加速后续的 Read 操作。
- Read->Invalid: Invalidate，即确保对应的 Cache 数据已经失效，且不影响内存数据。
- 其它：不作处理。

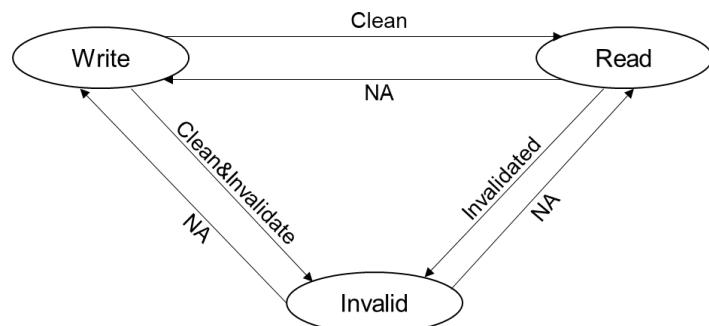


图 8-9 Ownership 状态转移图

遵循上述流程，可以保证共享内存的基本可使用。

## 8.2.7 死锁避免

### 8.2.7.1 概述

计算系统中存在语义依赖和资源依赖，如果处理不当，会导致出现死锁，系统停止工作。由于 UB 可支持多种组合场景，UB 芯片、软件设计以及开发人员应更加关注功能之间的耦合依赖，识别死锁的可能性。UB 提供了死锁避免的系统原则和机制，供设计人员使用，同时 UB 也支持设计人员自定义死锁避免机制。

内存访问以及消息通信是 UB 两种基础事务行为，两种行为中可能产生死锁的原因以及避免行为存在差异。本章节针对两种死锁避免行为展开描述。

### 8.2.7.2 内存访问中的死锁避免

UB 在单一物理端口上，提供多种应用场景的内存和 IO 访问功能。这些功能在访问内存对象（即内存段）时，可能会产生次生内存相关操作。原生操作和次生操作在执行过程中，如果形成流程依赖或者电路资源依赖，会导致系统死锁。

注：原生内存操作指处理器/加速器通过指令直接发起的内存操作，如 Load 指令而引起的总线读操作；次生内存操作指原生内存操作实现内存访问而触发的必要操作，如 Page Fault 处理，内存借用的 UMMU 处理等。

下面是三种典型死锁场景，供 UB 实现者参考。

- 内存池化借用

多 UBPU 平等工作时，UBPU 具有“内存使用方”和“内存提供方”双重角色，互为对方资源。Initiator 对 Target 的内存访问（下图中 Pooled Physical Memory Access），在 Target 转变为本地内存访问（下图中 Local Physical Memory Access）。两侧 UBPU 互相操作时，在 UB 实施的电路上，可能形成死锁依赖。

示例：节点 A 借用节点 B 的内存，节点 B 借用节点 A 的内存。当 A 和 B 同时通过 Writeback 更新对端内存，并且都需要等待对应的 TAACK 返回 Writeback 事务才认为操作结束。如果 TAACK 被反向的 Writeback 阻塞，则可能出现 TAACK 无法返回造成 Initiator 资源耗尽出现死锁问题。

- 页表访问

当内存访问中需要 UMMU 完成地址转换时，如果 UMMU 表项放到借用的内存上时，读取页表的操作也需要经过跟原始的 Memory 访问相同的端口，此时可能会产生死锁。并且，如果 UMMU 表项放到本节点的内存上时，读取页表的操作可能触发次生对原始 Memory 访问相同的端口，此时也可能会产生死锁。

示例：节点 A 节点借用节点 B 的内存，节点 A 向节点 B 发起内存操作时，节点 B 的 UMMU 页表在本节点内存上，节点 B 读取页表的响应如果触发对节点 A 的内存回写访问，而节点 A 正在等待节点 B 的内存访问，此时会导致双方都不能完成，形成死锁。

- Page Fault 处理

UB 内存访问支持内存动态管理。如果系统使能了内存动态管理，那么在访存路径上，可能产生 Page Fault。UBPU 需要完成 Page Fault 处理，才能继续完成 Memory 访问。Page Fault 的处理，可能产生对外部存储的次生访问( 下图中 Memory Hierarchy Swap )，也可能产生对其他 UBPU 内存的次生访问 ( 下图中 Pooled Physical Memory Access )，这些次生操作跟原始访存操作，经过共同电路，可能会导致系统死锁。

示例：节点 A 访问节点 B 的内存，如果节点 B 内存访问时出现 Page Fault，节点 B 需要处理 Page Fault，否则节点 A 的内存操作无法进行。当节点 B 迁移 Page 处理 Page Fault 时，可能造成 Page Fault 流量被节点 A 的内存访问流量阻塞，形成死锁。

UB 提供了请求重试、虚通道隔离、不同事务类型区分的机制，供实现者选择使用以处理此类场景问题。UB 实现者也可以通过保证页表本地化，且不依赖任何 UB 事务的无条件完成，来应对页表访问等死锁场景。

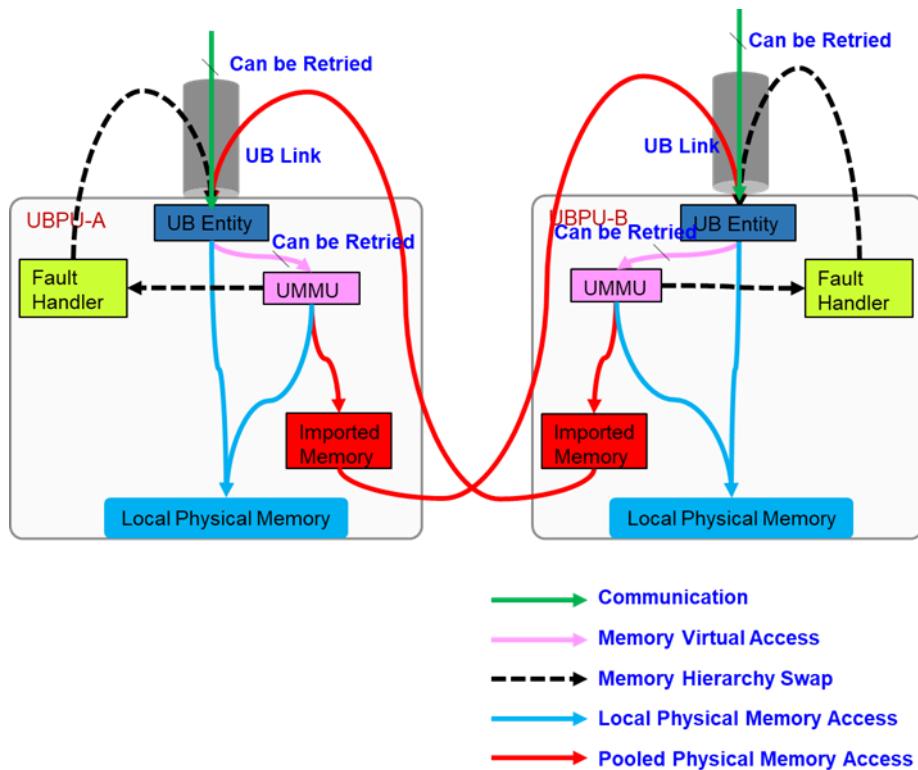


图 8-10 内存操作死锁避免

### 8.2.7.3 消息通信中的死锁避免

消息通信以队列的方式工作，当队列资源不足时，消息通信无法进行。不同 UBPU 上的消息队列，首尾相连，可能构成系统死锁的条件。场景如图 8-11 所示。

例如：节点 A 和节点 B 互相发送 Send 事务，由于可能有很多节点向节点 A 和 B 发起请求，导致 RQ 资源不足 ( Jetty 的多对多通信 )，此时节点 A 和 B 均会产生带有 RNR 的 TAACK。如果 TAACK 均被阻塞住 ( 如节点 A 发送 B 的 TAACK 被 Send 阻塞，节点 B 发往节点 A 的 TAACK 被 Send 阻塞 )，导致异常 TAACK 无法被处理，形成死锁。

UB 系统实现者可以通过资源预留规划、设定消息处理专属资源等方式，从系统维度消除死锁条件。同时，UB 也提供三种基础机制，帮助避免消息通信死锁。UB 实现者可综合选择利用。

- 传输层和事务层分离机制

消息事务处理过程中在功能层资源不足时，不会阻塞传输层以及之下协议层次的执行，避免单点的功能层资源不足，在电路上产生大范围的扩散反压。

- 事务层将资源状态返回事务响应，由发起侧重试

事务层返回不同响应状态，供发起侧重试或者采取其它策略，以此避免在电路上产生死锁等待。当 UB 事务层资源不足无法反馈响应时，支持通过传输层应答携带异常事务响应。

- 设置超时机制，允许消息通信失败，以释放资源

利用超时机制判定消息通信失败，并告知应用，由应用做进一步处理。UB 电路仍保持畅通。

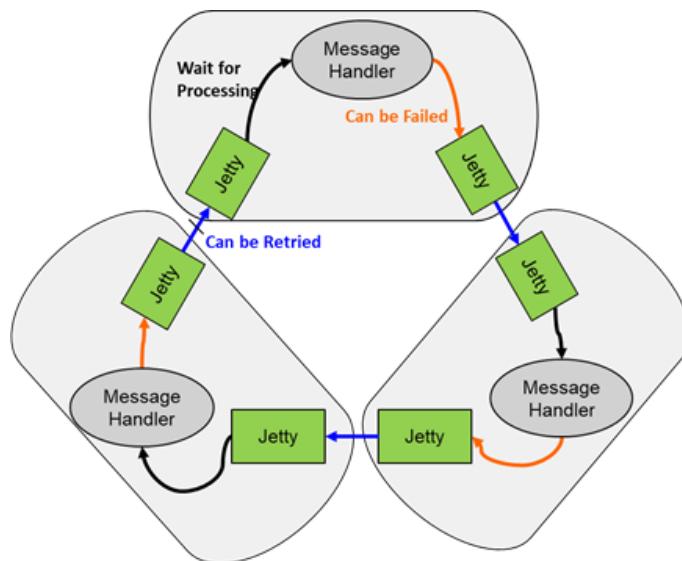


图 8-11 消息操作死锁避免

### 8.3 Load/Store 同步访问

本规范的 Load/Store 同步访问，泛指处理器指令集直接发出的各类操作行为。指令集操作本身以及它在工作中产生的次生事务，通过片上总线传递到 UB Controller，UB Controller 转换成 UB 事务层的各类事务操作。UB 不限定指令集架构，不同的指令集 Load/Store 行为，通过映射的 UB 事务层基础操作可实现互通。

UB 协议支持跨层协同优化以获得系统最优，所以，针对不同的端到端部署环境特征，UB Controller 可以采用适宜的事务层以及下层的特性组合完成工作。比如，针对服务器内、机架内、小范围集群等网络质量好的场景，Load/Store 指令的事务操作使用 TP Bypass 的工作模式，依赖数据链路层重传保障事务可靠传输，无需消耗传输层资源，优化时延以及带宽利用率。UB Controller 也支持将 Load/Store 指令扩展到数据中心范围，此时，传输层的 TP Channel 为 Load/Store 同步访问的事务提供端到端独立工作的可靠管道。

UB 也支持各指令集将 UB 事务作为原生组件，比如多对多消息发送、消息接收、事件通知、Order 信息传递、全局同步操作等，以支持多 UBPU 更加高效的协同工作。

另外，使用 Load/Store 同步访问调用事务层时，可使用事务层的 ROI，ROL 等事务服务模式（见 7.3.3 节）。

## 8.4 URMA 异步访问

开发者可使用 URMA 异步访问编程模型发起对 Target 的事务操作。Jetty 作为 URMA 基本通信单元，开发者可通过 Jetty 发起事务操作。

URMA 编程模型可采用事务队列方式接收并处理 Jetty 提交的事务请求，事务队列方式见 8.2.3。注意：事务队列是实现 URMA 异步访问的一种实现方式，也可使用更高效的交互方式。

开发者使用 Jetty 完成事务功能时，主要流程如下：

1. 获取 EID，并创建 URMA 上下文；
2. 基于 URMA 上下文，创建 Jetty、JFC 以及 JFCE 等，并建立通信关系，包括导入远端 Jetty 以及内存段等，创建过程中应绑定事务队列，并且指定选用的事务服务模式；
3. 开发者通过 Jetty 接口下发事务请求到 SQ 中。下发完事务请求后，可异步等待响应；
4. UB Controller 调度事务请求并转换为一次到多次的事务操作。用户下发的事务请求中应包含事务操作要求、Target 信息（内存段或 Target RQ 等）、保序要求、是否产生完成通知等；
5. 事务层根据事务操作类型以及要求完成事务操作；
6. 事务操作完成后，将完成信息上报到 CQ 中，如果完成后或者事务处理过程中出现异常等，需要产生事件通知，并需要上送到 EQ 中；
7. 开发者通过查询 CQ 绑定的 JFC 或者通过 Event 中断获得事务完成状态。如果出现异常，开发者根据需要进行异常处理。

UB Controller 根据事务请求调用事务层事务操作：

- 当调用内存事务时，UB Controller 依次调度 SQ 元素转换为内存事务（见 7.4.2 节），事务请求中应指定事务操作类型（读、写等）、要访问的内存地址、地址长度、和前序事务是否保证执行序等。
- 当调用消息事务时，UB Controller 依次调度 SQ 元素并完成发起消息操作（见 7.4.3 节），事务请求中应指定事务类型（Send 等）、要处理消息事务的 Target RQ 信息，执行序、完成序要求等。
- 当调用维护事务时，如果待更新的状态可通过直接修改内存更新，事务操作中需要指定内存信息；如果待更新的状态需要 Target 处理，需要指定要处理维护事务的 Target RQ 信息。

使用 URMA 异步访问调用事务层时，可使用事务层的 ROI，ROT，ROL，UNO 等事务服务。并不是每种请求都可使用上述四种服务模式，具体支持情况见事务操作处理流程（见 7.4 节）。

## 8.5 URPC

### 8.5.1 概述

URPC 是统一远程过程调用协议，其基于 UB 事务层提供的能力，支持任意 UBPU 之间直接发起平等函数调用。URPC 职能角色和协议流程如下：

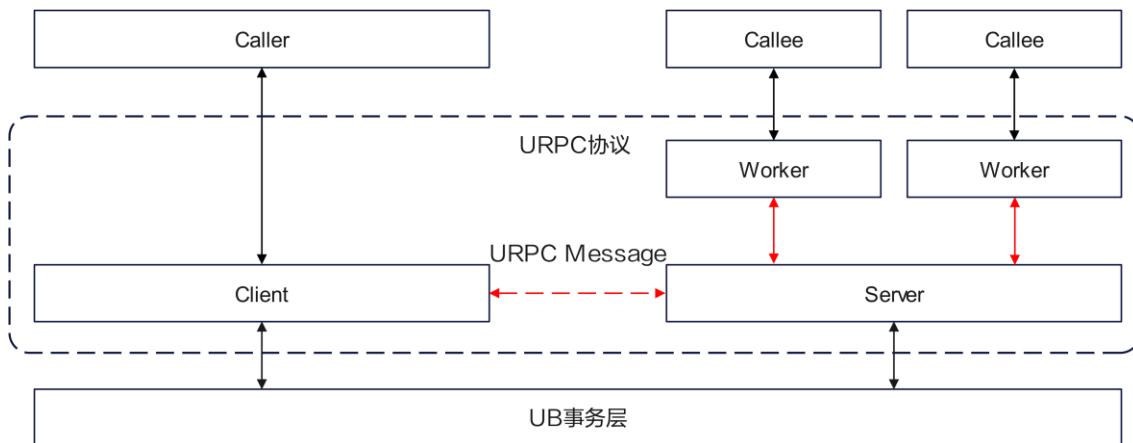


图 8-12 URPC 职能角色

URPC 包含 3 个职能角色：Client、Server、Worker，分别聚焦于不同的职责：

- Client：URPC 的发起端和调用者，负责向 Server 发起指定类型的远程函数调用；
- Server：URPC 的接收端和分配者，负责接收 Client 发起的远程函数调用并将其分配给某一 Worker 执行；
- Worker：URPC 的执行者，触发执行函数，在函数执行完毕后将结果返回 Server，再由 Server 返回 Client；

注 1：Caller：基于 URPC 协议发起远程过程调用的用户；

注 2：Callee：URPC 函数的具体实现方，根据诉求可与 Worker 合并；

在 Client 和 Server 之间通过 URPC Message 交互信息，分为以下类型：

- URPC Request：由 Client 发送给 Server，发起函数调用行为；
- URPC Ack：从 Server 返回 Client，代表参数传递完成，触发 Client 释放参数的内存空间；
- URPC Response：从 Server 返回 Client，代表函数执行完成并返回结果；

URPC Request 中携带标识用于在 Server/Worker 范围内唯一确定某一函数，URPC Message 和 Function 的详细格式定义见附录 H 节。

基于以上 URPC 角色和消息格式，URPC 协议流程如下：

- 用户 Caller 触发 Client 发起远程函数调用；
- Client 向 Server 发送 URPC Request 消息，携带函数唯一标识和参数信息；
- Server 接收到函数唯一标识和参数信息，向 Client 回复 URPC Ack 通知其参数传递完成；
- Server 将函数分配给某一 Worker 执行；

- Worker 基于函数唯一标识触发 Callee 执行对应函数，在函数执行完毕后将结果通过 URPC Response 返回给 Client；
- Client 收到 URPC Response，将远程函数调用结果返回给用户 Caller；

注 1：URPC Ack 是否需要由 Client 决定；

注 2：URPC Ack 与 URPC Response 是否合并由 Server 决定。如果合并，函数执行完毕后 Server 将结果通过 URPC Ack&Response 合并消息返回 Client，通知其参数传递完毕，同时一并将结果返回；

同时，URPC 协议支持以下创新特性：

- 平等函数调用：任意 UBPU 之间可以直接发起函数调用；
- 引用传参：支持 Worker 基于参数引用（参数数据地址）发起参数数据搬移；

### 8.5.2 平等函数调用

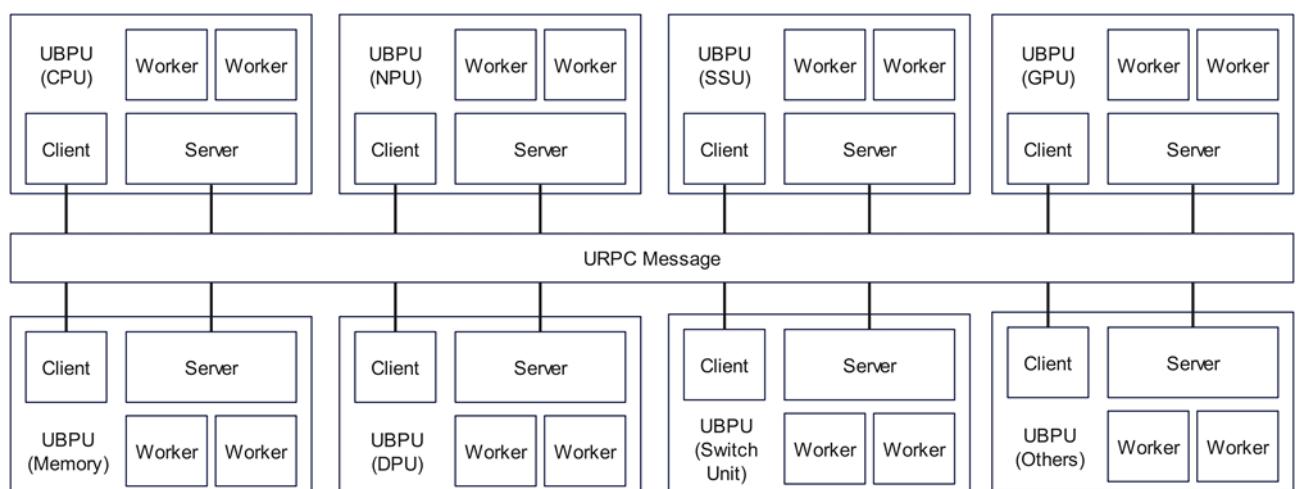


图 8-13 URPC 平等协议架构

基于 URPC 协议，Client/Server/Worker 可根据场景的诉求，实现为 UBPU 实体。受益于 UB 平等互访的架构设计，在任意 UBPU 上实现的 Client，都可以通过发送 URPC Message，向其它 UBPU 上实现的 Server/Worker 发起远程函数调用。典型应用如：UBPU (NPU) 直接向 UBPU (SSU) 发起远程存储写的函数调用，采用此方式，AI 训练或推理数据从 NPU 直接发到 SSU 上执行存储。

### 8.5.3 引用传参

URPC 支持如下三种参数传递方式：

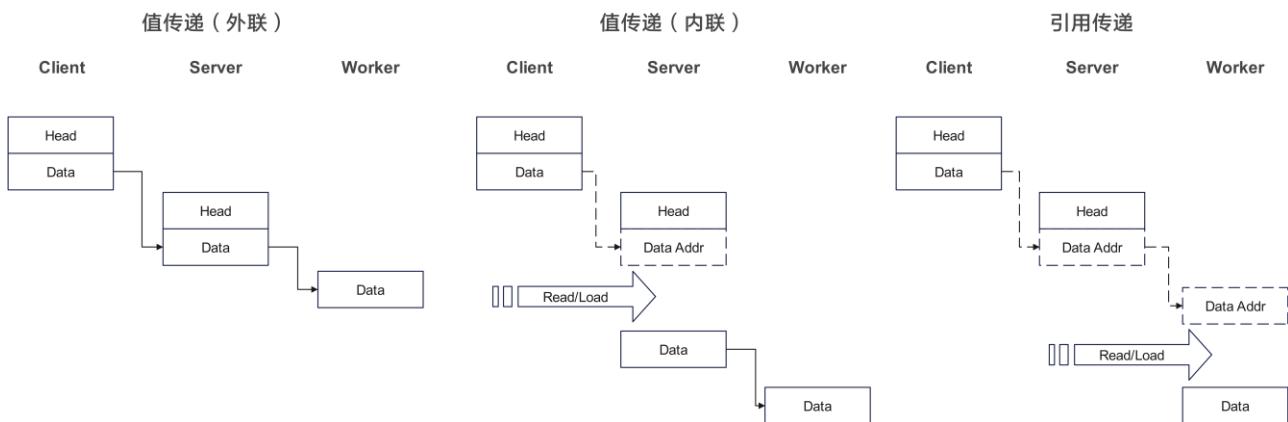


图 8-14 URPC 参数传递方式

- **值传递 (内联):** 参数数据和 URPC 协议头合并到一条 URPC Request 消息，由 Client 发送给 Server；
- **值传递 (外联):** 参数数据地址和 URPC 协议头合并到一条 URPC Request 消息，由 Client 发送给 Server，Server 在收到参数数据地址后通过 Read 或者 Load 语义向 Client 拉取完整的参数数据；
- **引用传递:** 参数数据地址和 URPC 协议头合并到一条 URPC Request 消息，由 Client 发送给 Server，Server 将参数数据地址传递给 Worker，由 Worker 通过 Read 或者 Load 语义向 Client 拉取完整的参数数据；

相较于值传递的方式，引用传递将参数传递的时机开放给 Worker 控制，从而让其有更大的灵活性协调参数传递和函数执行时机。

更多 URPC 参数传递方式的特点以及适用场景如下：

表 8-2 URPC 参数传递方式适用场景

参数传递方式	特点	适用场景
值传递 (内联)	<ul style="list-style-type: none"> <li>• 参数数据大小受 URPC Request 的大小上限约束；</li> <li>• 参数数据随 URPC Request 传递，参数传递需 0.5 个 RTT；</li> <li>• 参数数据从 Client 传递给 Server，再传递给 Worker</li> </ul>	参数数据小，内存资源充足 例如：存储场景，数据大小 40K 以下
值传递 (外联)	<ul style="list-style-type: none"> <li>• 参数数据大小不受 URPC Request 的大小上限约束；</li> <li>• Server 接收 URPC Request 后，再发起参数传输，参数传递需 1.5 个 RTT；</li> <li>• 参数数据从 Client 传递给 Server，再传递给 Worker</li> <li>• Server 拉取参数数据后，Worker 开始执行函数时，Client 即可释放参数的内存资源；</li> </ul>	参数数据大，内存资源不足 例如：存储场景，数据大小 40K 以上

参数传递方式	特点	适用场景
引用传递	<ul style="list-style-type: none"> <li>参数数据大小不受 URPC Request 的大小上限约束；</li> <li>Worker 开始执行函数后，发起参数传输，参数传递需 1.5 个 RTT；</li> <li>参数数据从 Client 直接传递给 Worker，传递时机由 Worker 灵活可控；</li> <li>Worker 执行函数拉取参数数据后，Client 才可释放参数的内存资源；</li> </ul>	<p>参数传递和函数执行时间较接近 例如：AI 训练/推理场景，数据传输和 NPU 运算需相互掩盖执行时间</p>

注：RTT ( Round-Trip Time ) 表示往返时间

## 8.6 多 Entity 协同

UB 在事务类型、访问控制、传输方式和资源访问等方面实现了多 UBPU 之间的统一，显著提升了多 UBPU 协同完成任务的效率。UB 芯片与软件的实施者能够灵活组合事务层机制，定义符合场景需求的多 Entity 协同。协同操作通过单次功能调用发起，由 UB 体系将其分解并映射为多个底层 UB 事务，最终在一个或多个 UBPU 上协同执行。

以下为几种典型的协同操作场景示例：

### 1. 融合操作

通过将多个离散的事务组合并定义为统一的融合操作。该操作单次可处理更多操作数或实现更复杂的操作逻辑，有效缓解软件同步开销和单线程性能瓶颈。典型的融合操作包括多 UBPU 的广播操作、组播操作、任务均衡操作、任务调度操作、数据与同步复合操作等。

### 2. 集合通信操作

集合通信是并行计算中的经典范式。通过定义协同操作，一次集合通信操作可被抽象为单一的功能调用。该调用可根据实现被分解为多个 UB 事务，并在多 UBPU 并行环境中，将分解后的事务动态映射至各 UBPU 协同执行，从而有效减少数据移动次数，提升系统同步效率。

### 3. 全局维护操作

通过单次功能调用，可实现跨多个 UBPU 上的 Memory 一致性维护、UMMU 变更同步、通信状态管理等系统级维护功能。

UB 功能层定义了模块化协同操作功能的框架，允许将新的场景化设计集成至 UB 体系中，提升 UB 系统整体性能。

## 8.7 Entity 管理

UBPU 是 Entity 资源的使用者，除了实现对 Entity 所声明的功能或服务的调用，还应实现对 Entity 的相关管理，包括本地 Entity 发现、池化 Entity 注册、配置管理、中断与消息通知、通信与远端内存注册控制、虚拟化等管理功能，详见资源管理章节。

# 9 内存管理

## 9.1 概述

为支持 UB 总线上 UBPU 间内存资源的共享，以及确保对内存的合法访问，UB 系统中的内存资源需要被高效的管理起来（分配、授权和回收），从而支持用户安全、动态、高效的使用 UB 内存资源。为达成上述目标，本章所述内存管理设计了如下关键特性：

1. 定义全局统一的 UB 内存描述符（UB Memory Descriptor, UBMD），用于描述 UB 系统中的内存段。支持多方通过前文功能层所述同步/异步的方式基于 UBMD 使用内存段。
2. 基于内存管理者和内存使用者定义 Home-User 访问模型。Home 是内存物理资源的拥有者，User 是内存的访问者。User 基于 UBMD 访问 Home 的内存/数据。
3. 设计 UB Memory Management Unit ( UMMU ) 作为 UB 系统中的内存访问控制组件，在 Home 侧提供内存资源的地址映射和访问权限校验。UB 基于 Home 侧 UMMU 进行访问控制，UMMU 可将权限校验安全下放到非特权级软件。
4. 设计 UB Decoder 作为 UB 系统中 User 侧 UBPU 对 UBMD 访问的入口。在 User 侧按需将 UBPU 侧本地地址转换为 UBMD 发送给 Home。

## 9.2 Home-User 访问模型

Home-User 访问模型如图 9-1 所示：

1. User 向 Home 发起内存访问请求，该请求应包含 UBMD。User 发给 Home 的 UBMD 可来自于：
  - (1) 由 UB Decoder 基于 PA 查表得到，见 8.3 节。
  - (2) 由用户通过 User 侧编程接口提供，见 8.4 节。
2. Home 接收并处理内存访问请求。Home 侧使用 UMMU 将 UBMD 转换为目标内存的 Physical Address ( PA )，并进行权限校验。

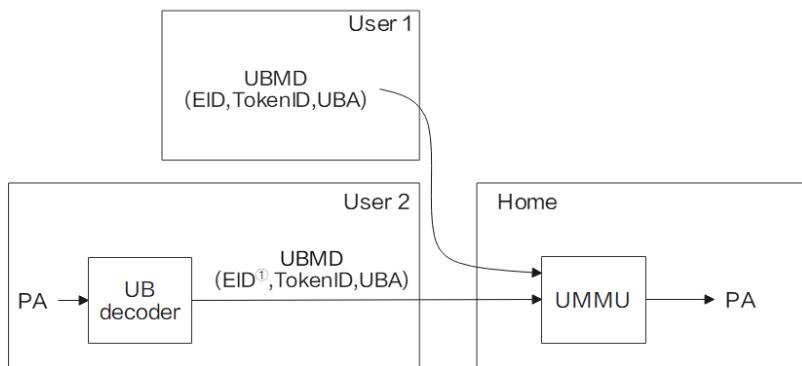


图 9-1 Home-User 访问模型

注①：Home-User 访问模型提供了一种简化模式。该模式下，内存访问请求无需携带 EID，Home 侧 UMMU 的处理流程也会做对应的简化处理。

### 9.3 UB 内存描述符

UBMD 包括 EID、TokenID 和 UB Address ( UBA )，用于索引 Home 侧的 PA。EID 标识目标内存所在的 Entity。TokenID 标识目标内存所属的 UBA 空间和 UBA 空间的权限集合。UBA 是 Home 提供给 User 的虚拟地址，用于访问 Home 侧的内存资源，位宽为 64 bits。

UBA 空间是 UBA 的集合，可映射至 Home 侧 PA 空间，UBA 和 PA 的映射关系存放于内存地址转换表 ( Memory Address Translation Table , MATT ) 中。UBA 空间和权限集合的映射关系存放于内存地址权限表 ( Memory Address Permission Table , MAPT ) 中。地址转换和权限校验均使用 UBA 作为索引，分别对 MATT 和 MAPT 进行查找，最终获得 PA 和权限信息，如图 9-2 所示。

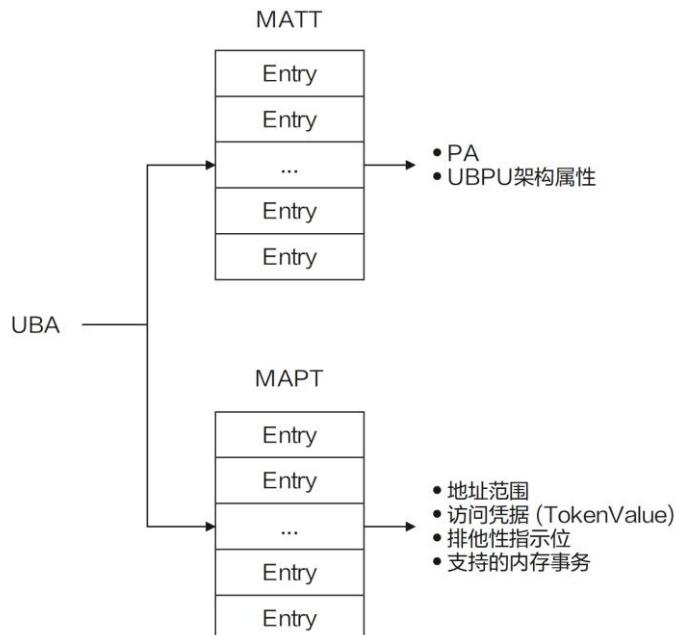


图 9-2 以 UBA 作为索引查找 MATT 和 MAPT

### 9.4 UMMU 功能与处理流程

#### 9.4.1 一般要求

内存访问的处理流程中，UMMU 将访问请求携带的 UBMD 作为输入，获得内存 PA 和权限信息。

UMMU 对内存访问的处理流程如图 9-3 所示：

1. 配置查找：UMMU 根据 EID 查询 Target Entity Configuration Table ( TECT )，获取 Target Context Table ( TCT ) 基地址。

2. 上下文查找：UMMU 根据 TokenID 查询 TCT，获取 MATT 基地址和 MAPT 基地址。
3. 地址转换：UMMU 根据 UBA 查询 MATT，获取目标内存 PA。
4. 权限校验：UMMU 根据 UBA 查询 MAPT，获取目标内存的权限信息，判断此次内存访问是否合法。

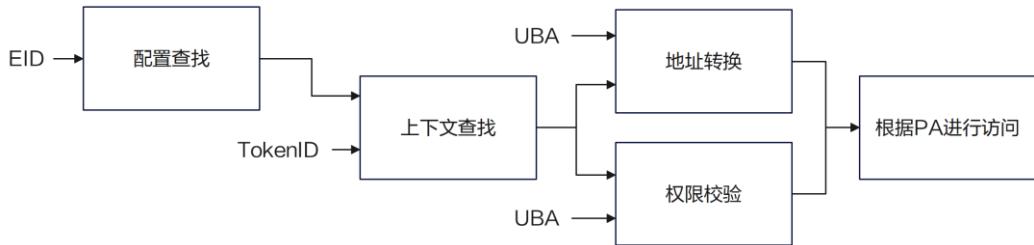


图 9-3 访问处理流程示意

## 9.4.2 配置查找和上下文查找

### 9.4.2.1 配置查找

UMMU 以 Home 侧 Entity 为粒度进行配置。配置信息存放在 TECT 中，包含如下内容：

1. 后续流程启用情况：是否启用权限校验、两阶段地址转换、事件上报合并。
2. 内存访问属性：内存属性、内存属性选择机制。
3. 上下文查找和地址转换流程信息。
4. 流量监控标识。

在处理内存访问的流程中，UMMU 根据 EID 查询 TECT，从 TECT 中找到对应表项（即 Target Entity Configuration Table Entry，TECTE），以获取配置信息。配置查找如图 9-4 所示。

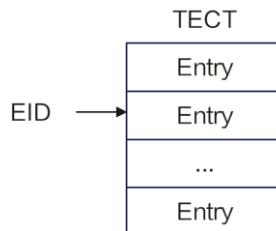


图 9-4 配置查找示意

TECTE 的数据结构格式如图 9-5 所示。下文中标记为 IMPL\_DEF 的位域（段）为协议开放给具体的实现可自行定义的部分。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
3~0	IMPL_DEF												EM_EN	MAPT_EN	INST_SEL	PRIV_SEL	SECURE_SEL	IMPL_DEF	MEM_ATTR_SEL	MEM_ATTR				ST_MODE			V																					
7~4	RSVD												S2_VMID[15:0]																																			
11~8	TCT_PTR[31:6]																															TCT_NUM																
15~12	IMPL_DEF												TCT_FMT	TCT_PTR[51:32]																																		
19~16	IMPL_DEF												S2_FBR	S2_FBS	S2_AFFD	S2_HAF	S2_HDF	S2_ENDIANI	S2_PAS	IMPL_DEF	NS_S2_TG	NS_S2_SL	S_S2_TSZ																									
23~20	IMPL_DEF	RSVD																												RSVD																		
27~24	NS_S2_MATT[31:4]																														RSVD																	
31~28	RSVD												NS_S2_MATT[51:32]															RSVD																				
35~32	S_S2_MATT[31:4]																															RSVD																
39~36	RSVD												S_S2_MATT[51:32]															RSVD																				
43~40	MTMC_PTR[31:12]																															RSVD																
47~44	RSVD												MTMC_PTR[51:32]															RSVD																				
51~48	RSVD												IMPL_DEF												MTM_ID[15:0]															RSVD								
55~52	RSVD																																RSVD															
59~56	RSVD																																RSVD															
63~60	RSVD																																RSVD															

图 9-5 目标实例配置表表项

TECTE 数据结构格式中的各域段含义见表 9-1。

表 9-1 目标实例配置表表项域段描述

域段名称	位宽 ( Bits )	描述
V	1	Valid, 指示当前 TECTE 是否有效。 1'b0: 无效, 此时忽略 TECTE 中其它域段信息; 1'b1: 有效。 若内存访问命中的 TECTE.V==1'b0, 则本次内存访问将被终止, 向 User 返回错误信息, 并上报事件。
ST_MODE	3	Stage Mode, 指示地址转换 Stage 1 和 Stage 2 的启用情况。 3'b000~3'b100: 无效取值, 直接向 User 返回错误信息, 但

域段名称	位宽 ( Bits )	描述
		<p>UMMU 不上报事件。</p> <p>3'b101: 仅启用 Stage 1;</p> <p>3'b110: 仅启用 Stage 2;</p> <p>3'b111: Stage 1 和 Stage 2 均启用。</p> <p>若 UMMU 不支持 Stage 1, 则取值 3'b1x1 非法;</p> <p>若 UMMU 不支持 Stage 2, 则取值 3'b11x 非法;</p> <p>若 UMMU 不支持安全态下的 Stage 2, 则安全态的 TECTE 取值 3'b11x 非法。</p> <p>当出现非法取值时, 本次内存访问将被终止, 向 User 返回错误信息, 并上报事件。</p>
MEM_ATTR	4	<p>Memory Attribute, 内存属性覆盖值。</p> <p>如果 MEM_ATTR_SEL==1'b1, 则使用本域段提供的 memory 属性作为处理访问请求时访问目标内存的属性。</p> <p>注: MEM_ATTR 在协议中不定义其具体 bit 含义, 此处遵循 UMMU 所在系统架构中对内存属性的定义。</p>
MEM_ATTR_SEL	1	<p>Memory Attribute Select, 指示 Memory attribute 选择机制。</p> <p>1'b0: 使用当前处理的内存访问请求提供的内存属性;</p> <p>1'b1: 使用 TECTE.MEM_ATTR 提供的属性。</p>
SECURE_SEL	2	<p>Secure Select, 指示 Secure 属性的选择机制。</p> <p>2'b00~2'b01: 使用当前处理的内存访问请求提供的 Non-secure attribute。</p> <p>2'b10~2'b11: 使用本域段配置的值来选择 Secure 属性, SECURE_SEL[0]含义如下:</p> <ul style="list-style-type: none"> <li>• 1'b0: Secure;</li> <li>• 1'b1: Non-secure。</li> </ul>
PRIV_SEL	2	<p>Privilege Select, 指示 Unprivileged/Privileged 属性的选择机制。</p> <p>2'b00~2'b01: 使用当前处理的内存访问请求提供的 Unprivileged/Privileged 属性;</p> <p>2'b10~2'b11: 使用本域段配置的值来覆盖 Unprivileged/Privileged 属性, PRIV_SEL[0]含义如下:</p> <ul style="list-style-type: none"> <li>• 1'b0: Unprivileged;</li> <li>• 1'b1: Privileged。</li> </ul>

域段名称	位宽 ( Bits )	描述
INST_SEL	2	<p>Instruction/Data Select, 指示指令/数据属性的选择机制。</p> <p>2'b00~2'b01: 使用当前处理的内存访问请求提供的数据/指令属性；</p> <p>2'b10~2'b11: 使用本域段配置的值来覆盖数据/指令属性， INST_SEL[0]含义如下：</p> <ul style="list-style-type: none"> <li>• 1'b0: 数据；</li> <li>• 1'b1: 指令。</li> </ul>
MAPT_EN	1	<p>MAPT Enable, 表示当前 TECTE 中 MAPT 的启用情况, 即索引到当前 TECTE 的内存访问是否可以使用 MAPT 进行权限校验：</p> <p>1'b0: 不使用 MAPT 进行权限校验, 此时 TCT 中任意 TCTE 均不启用 MAPT；</p> <p>1'b1: 可使用 MAPT 进行权限校验, 此时 TCT 中各个 TCTE 可独立配置 MAPT 的启用情况。</p>
EM_EN	1	<p>Event Merge Enable, 表示事件合并的启用情况。</p> <p>1'b0: 不允许对事件进行合并；</p> <p>1'b1: 允许对类似事件进行合并。</p> <p>UMMU 能够通过合并共享相同页面颗粒的地址、访问类型和 TokenID 的故障记录来减少事件队列的使用。</p>
S2_VMID	16	Stage 2 Virtual Machine Identifier, 用于区分 TLB entries 所对应的虚拟机。
TCT_NUM	5	<p>TCT Number, 表示 TCT_PTR 所指向的 TCTE 数量, 具体为 <math>2^{TCT\_NUM}</math>。</p> <p>若 Stage 1 未启用, 则该域段会被忽略。</p> <p>若 Stage 1 启用, 若 TCT_NUM==0, 则该内存访问将被终止, 并上报事件。当 TCT_NUM&gt;0 时, 若内存访问提供的 TokenID 超出此域段指定的范围, 则内存访问将被终止, 并上报事件。</p>
TCT_PTR	46	<p>TCT Pointer, Stage 1 对应的 TCT 信息所在位置指针, 该域段为地址指针的 TCT_PTR[51:6]。默认地址 64 Bytes 对齐。</p> <p>如果 Stage 1 未启用, 则该域段被忽略。</p> <p>如果启用 Stage 2, 若该域段表示 Intermediate Physical Address ( IPA ), 则该域段范围不能超过 Intermediate Physical Address Size ( IPAS ), 若该域段表示 Physical Address ( PA ), 则其范围不能超过 Physical Address Size ( PAS )。</p>
TCT_FMT	2	TCT Format, UMMU 上下文查找启用时, 所用的 TCT 的组织形式。 2'b00: 线性 TCT, 使用 TokenID[TECTE.TCT_NUM-1:0]进行索引；

域段名称	位宽 ( Bits )	描述
		2'b01: 两级 TCT，其中 L2 TCT 所包含表项数量为 64; 2'b10~2'b11: 保留。 当支持并启用多个 TokenID 时，支持的范围为 $2 \sim 2^{20}$ 个 TokenID。 若 Stage 1 未启用，则该域段被忽略。
TCT_STALL_EN	1	TCT Stall Enable，表示当芯片支持 Stall 能力时，是否启用事件上报。 1'b0: 不启用事件上报； 1'b1: 启用事件上报。
TCT_MTM_EN	1	TCT Memory Traffic Monitoring Enable，指示是否启用 TCT Memory Traffic Monitoring。 1'b0: 不启用，当前处理的内存访问请求的 MTM_ID 和 MTM_GP 来自于 TECT.MTM_ID 和 TECT.MTM_GP； 1'b1: 启用，当前处理的内存访问请求的 MTM_ID 和 MTM_GP 来自于 TCT.MTM_ID 和 TCT.MTM_GP。
NS_S2_TSZ	6	Non-Secure Stage 2 Table Size，表示非安全态 Stage 2 地址转换表涵盖的 IPA 输入区域的大小。 注 1：具体取值对应的大小协议不做定义，由系统架构定义。 注 2：当 Stage 2 未启用时，该域段保留。
NS_S2_SL	2	Non-Secure Stage 2 Starting Level，表示非安全态 Stage 2 地址转换表的 Starting Level。 注 1：具体取值对应的 Starting Level 协议不做定义，由系统架构定义。 注 2：当 Stage 2 未启用时，该域段保留。
NS_S2_TG	2	Non-Secure Stage 2 Translation Granule，表示非安全态 Stage 2 地址转换表的转换粒度。 注 1：具体取值对应的转换粒度协议不做定义，由系统架构定义。 注 2：当 Stage 2 未启用时，该域段保留。
S2_PAS	3	Stage 2 Physical Address Size，指示 Stage 2 输出的 PA 地址位宽。 3'b000: 32 bits； 3'b101: 48 bits； 其它：保留。
S2_ENDIAN	1	Stage 2 Endianness，指示 Stage 2 地址转换表的大小端。 1'b0: 小端； 1'b1: 大端； 当 Stage 2 未启用时，该域段保留。

域段名称	位宽 ( Bits )	描述
S2_HDF	1	Stage 2 Hardware Dirty Flag, 指示硬件是否可以自动更新 Stage 2 地址转换表中的“Dirty”标识位。 具体含义参考 S2_HAF 域段描述。
S2_HAF	1	Stage 2 Hardware Access Flag, 指示硬件是否可以自动更新 Stage 2 地址转换表中的“Access”标识位。 {S2_HDF, S2_HAF}组合取值含义说明： 2'b00: 不启用硬件自动更新标识位的功能； 2'b01: 启用硬件更新“Access”标识位； 2'b10: 保留位； 2'b11: 启用硬件更新“Access”和“Dirty”标识位。
S2_AFFD	1	Stage 2 Access Flag Fault Disable, 指示是否禁用 Stage 2“Access”标识位故障上报。 当硬件未启用“Access”标志位自动更新功能时，若访问到 MATTE.AF=1'b0 的 MATTE，行为如下： 1'b0: “Access”标识位上报故障，其处理行为由 TECTE.S2_FBS 和 TECTE.S2_FBR 决定； 1'b1: “Access”标识位不上报故障。 当 S2_HAF==1'b1，该域段被忽略。 当 Stage 2 未启用时，该域段保留。
S2_FBS	1	Stage 2 Fault Behavior Stall, 指示 Stage 2 地址转换出错后处理行为为 Stall。 当 Stage 2 未启用时，该域段保留。
S2_FBR	1	Stage 2 Fault Behavior Record, 指示 Stage 2 地址转换出错后处理行为为 Record。 当 Stage 2 未启用时，该域段保留。
S_S2_TSZ	6	Secure Stage 2 Table Size, 表示安全态 Stage 2 地址转换表涵盖的 IPA 输入区域的大小。 注 1：具体取值对应的大小协议不做定义，由系统架构定义。 注 2：当 Stage 2 未启用时，该域段保留。
S_S2_SL	2	Secure Stage 2 Starting Level, 表示安全态 Stage 2 地址转换表的 starting level。 注 1：具体取值对应的 Starting Level 协议不做定义，由系统架构定义。 注 2：当 Stage 2 未启用时，该域段保留。

域段名称	位宽 ( Bits )	描述
S_S2_TG	2	Secure Stage 2 Translation Granule, 表示安全态 Stage 2 地址转换表的转换粒度。 注 1: 具体取值对应的转换粒度协议不做定义, 由系统架构定义。 注 2: 当 Stage 2 未启用时, 该域段保留。
NS_S2_MATT	48	Non-Secure Stage 2 MATT, 表示非安全态 Stage 2 地址转换表入口地址, 本域段为 NS_S2_MATT[51:4]。默认地址 16 Bytes 对齐。 当 Stage 2 未启用时, 该域段保留。
S_S2_MATT	48	Secure Stage 2 MATT, 表示安全态 Stage 2 地址转换表入口地址, 本域段为 S_S2_MATT[51:4]。默认地址 16 Bytes 对齐。 当 Stage 2 未启用时, 该域段保留。
MTMC_PTR	40	Memory Traffic Monitoring Context Pointer, 本域段为 MTMC_PTR [51:12]。默认地址 4KB 对齐。
MTM_ID	16	Memory Traffic Monitoring ID。
MTM_GP	8	Memory Traffic Monitoring Group。

#### 9.4.2.2 上下文查找

##### 9.4.2.2.1 一般要求

UMMU 经过配置查找获得 TECTE 后, 通过其中的 ST\_MODE 域段获得对应 Entity 的 Stage 1 地址转换启用情况。

当 Entity 启用 Stage 1 地址转换时, UMMU 根据 TokenID 查询 TCT, 获取所访问内存段的上下文信息, 包括 Stage 1 地址转换流程相关信息、权限校验启用情况等。TCT 的基地址记录在 TECTE.TCT\_Ptr 域段。TCT 包含线性 TCT 和两级 TCT 两种形式, 由 TECTE.TCT\_FMT 域段决定。

当 Entity 未启用 Stage 1 地址转换时, UMMU 不执行上下文查找。

##### 9.4.2.2.2 线性 TCT

线性 TCT 由若干个连续的 TCTE 组成, 使用 TokenID 直接进行索引 (如图 9-6 所示)。

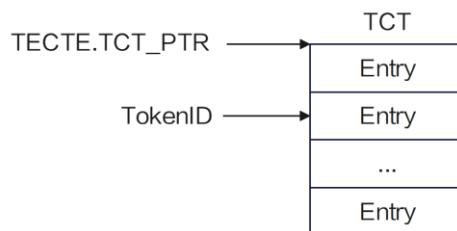


图 9-6 线性 TCT 结构和查找流程示意

TCTE 的数据结构格式如图 9-7 所示。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																								
3~0	RSVD										E_Bit	MAC_EN	MAPT_EN	RTGS		MAPT_MODE	IMPL_DEF	FBR	FBS	HAF	HDF	AFFD	IMPL_DEF	GPAS		ENDI	IMPL_DEF	V																																												
7~4	RSVD										IMPL_DEF																																																													
11~8	RSVD										IMPL_DEF										MATTWD	TGS	SZ																																																	
15~12	RSVD				MTM_GP				MTM_ID																		IMPL_DEF																																													
19~16	MATTBA[31:4]										IMPL_DEF																		IMPL_DEF																																											
23~20	IMPL_DEF	NS	RSVD								MATTBA[51:32]																		IMPL_DEF																																											
27~24	MAPT_BBA[31:5]																		IMPL_DEF						MAPT_BB_MA																																															
31~28	MAPT_BB_SZ	RSVD								MAPT_BBA[47:32]																		IMPL_DEF																																												
35~32	MAPT_BTA[31:5]																		IMPL_DEF						MAPT_BT_MA																																															
39~36	RSVD	MAPT_BT_SZ	RSVD								MAPT_BTA[47:32]																		IMPL_DEF																																											
43~40	MATTR0[31:0]																																																																							
47~44	MATTR1[31:0]																																																																							
51~48	RSVD																																																																							
55~52	RSVD																																																																							
59~56	RSVD																																																																							
63~60	RSVD																																																																							

图 9-7 目标上下文表项

TCTE 数据结构中各域段含义见表 9-2。

表 9-2 目标上下文表项域段描述

域段名称	位宽 ( Bits )	描述
V	1	Valid, 指示当前 TCT Entry 是否有效。 1'b0: 当前 TCTE 无效; 1'b1: 当前 TCTE 有效。 当上下文查找获得的 TCTE.V==1'b0 时, 需要终止当前内存访问, 并上报事件。

域段名称	位宽 ( Bits )	描述
ENDI	1	<p>Endianness, 指示 Stage 1 地址转换表和 Stage 1 权限表的大小端模式。</p> <p>1'b0: 小端模式； 1'b1: 大端模式。</p> <p>注：如果 TCTE.MATTWD==1'b1，该域段保留。</p>
GPAS	3	<p>Guest Physical Address Size, 指示 MATT 和 MAPT 基地址（通常为 PA）的位宽：</p> <p>3'b000: 32 bits 3'b001: 36 bits 3'b010: 40 bits 3'b011: 42 bits 3'b100: 44 bits 3'b101: 48 bits 3'b110~3'b111: 保留</p> <p>注：Stage 1 地址转换完成后通过 GPAS 来检查输出地址范围，如果超出范围则会上报事件。如果[51:eff_GPAS]不为 0 则上报事件，其中 eff_GPAS 为该域段表示实际位宽。</p>
AFFD	1	<p>Access Flag Fault Disable, 指示是否禁用 Stage 1 “Access” 标识位故障上报。</p> <p>当硬件未启用“Access”标识位自动更新功能时，若访问到 MATTE.AF==1'b0 的 MATTE 时，行为如下：</p> <p>1'b0: “Access” 标识位上报故障，其处理行为由 TCTE.FBS 和 TCTE.FBR 决定； 1'b1: “Access” 标识位不上报故障。</p> <p>注：TCTE.HAF==1'b1 时，该域段被忽略。</p>
HDF	1	<p>Hardware Dirty Flag, 指示硬件是否可以自动更新 Stage 1 地址转换表中的“Dirty”标识位。</p> <p>具体含义参考 HAF 域段描述。</p>
HAF	1	<p>Hardware Access Flag, 指示硬件是否可以自动更新 Stage 1 地址转换表中的“Access”标识位。</p> <p>{HDF, HAF}组合取值含义说明：</p> <p>2'b00: 不启用硬件自动更新标识位的功能； 2'b01: 启用硬件更新“Access”标识位； 2'b10: 保留； 2'b11: 启用硬件更新“Access”和“Dirty”标识位。</p>

域段名称	位宽 ( Bits )	描述
FBS	1	Fault Behavior Stall, 指示 Stage 1 地址转换出错后处理行为为 Stall。 若 TECTE.TCT_STALL_EN==1'b1, FBS 配置为 1 则为非法配置。
FBR	1	Fault Behavior Record, 指示 Stage 1 地址转换出错后处理行为为 Record。
MAPT_MODE	1	MAPT Mode, 指示当前 TCTE 包含的 MAPT 的组织形式： 1'b0: 单表项 MAPT; 1'b1: 多级 MAPT。
RTGS	2	Range Table Granule Size, 该域段在多级 MAPT 下有效, 表示 MAPT 最末级可表达的最大粒度： 2'b00: 保留; 2'b01: 粒度为 4KB; 2'b10: 粒度为 2MB; 2'b11: 保留。
MAPT_EN	1	MAPT Enable, 指示当前 TCTE 中 MAPT 的启用情况, 即索引到当前 TCTE 的内存访问是否需要使用 MAPT 进行权限校验： 1'b0: 不使用 MAPT 进行权限校验; 1'b1: 使用 MAPT 进行权限校验。 注: 当 TECTE.MAPT_EN==1'b0 时, 该域段被忽略。
MAC_EN	1	MAC Enable, 表示当前 TokenValue 校验模式： 1'b0: 随机数模式; 1'b1: 保留。
E_Bit	1	Exclusive-Bit, 校验规则见节 9.4.4.3.3。 当 E_Bit 校验失败时, 需上报事件。
SZ	6	Size, UBA 的最大位宽, 位宽数值为 64-SZ。 注: SZ 的有效范围由寄存器和所在系统架构决定。
TGS	2	Translation Granule Size, 指示当前地址转换表的粒度。 注: 具体取值对应的转换粒度协议不做定义, 由系统架构定义。
MATTWD	1	MATT Walk Disable, 指示是否禁用 MATT 进行地址转换： 1'b0: 使用 MATT 进行地址转换; 1'b1: 不使用 MATT 进行地址转换。TLB 未命中时上报事件, 此时 TCTE.SZ/TGS/TTBA 域段将被忽略。
MTM_ID	16	Memory Traffic Monitoring ID

域段名称	位宽 ( Bits )	描述
		当 TECTE.TCT_MTM_EN==1'b0 时，忽略该域段。
MTM_GP	8	Memory Traffic Monitoring Group 当 TECTE.TCT_MTM_EN==1'b0 时，忽略该域段。
MATTBA	48	Adderss of MATT Base, bits[51:4] TECTE.MATTWD==1'b1 时，该域段忽略。
NS	1	Non-Secure，指示访问 MATT 指向的 starting level 转换表的 memory Non-Secure 属性： 1'b0: Non-Secure； 1'b1: Secure。 注：该域段只有当从安全 TECTE 访问时使用，否则忽略该域段。
MAPT_BB_MA	2	MAPT Base Block Memory Attribute， MAPT Base Block 的内存属性，取值由所在系统决定。
MAPT_BBA	43	MAPT Base Block Address，表示 MAPT Base Block 的基地址中的[47:5]位。 MAPT_BBA 地址范围不能超过 GPAS 定义的范围,否则为非法配置，并上报事件。
MAPT_BB_SZ	4	MAPT Base Block Size，取值范围是 4'b0000~4'b1001，其他取值非法。表示 MAPT Base Block 的大小，具体为 4KB * $2^{\text{MAPT_BB_SZ}}$ 。 在 TCTE.MAPT_MODE==1'b0 时，忽略该域段。
MAPT_BT_MA	2	MAPT Block Table Memory Attribute， MAPT Block Table 的内存属性，取值由所在系统决定。
MAPT_BTA	43	MAPT Block Table Address，表示 MAPT Block Table 的基地址中的[47:5]位（其余位为 0）。 MAPT_BTA 地址范围不能超过 GPAS 定义的范围，否则为非法配置，并上报事件。
MAPT_BT_SZ	3	MAPT Block Table Size，表示 MAPT Block Table 的大小，具体为 4KB * $2^{\text{MAPT_BT_SZ}}$ ，最小为 4KB，最大取值为 512KB。 在 MAPT_MODE==1'b0 时，忽略该域段。
MATTR0	32	Memory Attribute 0 Memory 属性 0，指示查得页表的属性，通过地址转换页表中的 AttrIndx 域段来选择属性。 注：协议中不定义 MATTR0 具体 bit 含义，此处遵循 UMMU 所在系统架构。

域段名称	位宽 ( Bits )	描述
MATTR1	32	Memory Attribute 1 Memory 属性 1，指示查得页表的属性，通过地址转换页表中的 AttrIndx 域段来选择属性。 注：协议中不定义 MATTR1 具体 bit 含义，此处遵循 UMMU 所在系统架构。

#### 9.4.2.2.3 两级 TCT

当采用两级 TCT 时，依照 UMMU 查找的先后顺序，应将相关数据结构分为 Level 1 Target Context Table ( L1 TCT ) 和 Level 2 Target Context Table ( L2 TCT )。L1 TCT 由若干个指向 L2 TCT 的指针组成。

使用两级 TCT 的 UMMU 上下文查找流程如图 9-8 所示：

- 首先使用 TokenID[TCT\_NUM-1:6] 作为索引对 L1 TCT 进行查找，从命中的表项(即 L1 TCTE) 中获得 L2 TCT 的基地址；
- 再使用 TokenID[5:0]作为索引对 L2 TCT 进行查找，从命中的表项 (即 L2 TCTE) 中获得上下文信息。

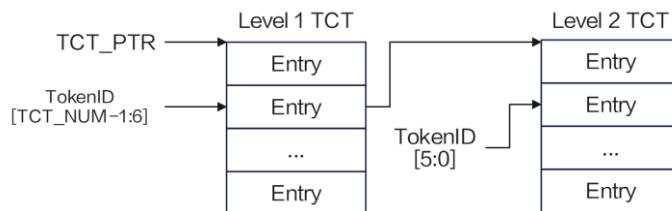


图 9-8 两级 TCT 结构和查找流程示意

L1 TCTE 数据结构格式如图 9-9 所示。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
3~0																															V	
7~4																															RSVD	

图 9-9 Level 1 目标上下文表表项

L1 TCTE 数据结构格式中各域段含义见表 9-3。

表 9-3 L1 目标上下文表表项域段描述

域段名称	位宽 ( Bits )	描述
V	1	Valid, 指示当前 L1 TCTE 是否有效。 1'b0: 无效，此时忽略其它域段信息； 1'b1: 有效。
L2TCTPtr	40	L2 TCT Pointer, 表示 L2 TCT 的基地址中的[51:12]位。 若启用 Stage 2，则该域段配置值不能超过 IPAS；若未启用 Stage 2，则该域段配置值不能超过 PAS。

L2 TCTE 的数据结构格式和域段含义与 L1 TCTE 相同。

### 9.4.3 UMMU 地址转换

UMMU 经过配置查找和上下文查找后，获得了 UBA 空间的上下文信息。UMMU 通过地址转换流程，将 UBA 转换为 PA。UMMU 支持虚拟化场景下的两阶段地址转换，整体功能流程如图 9-10 所示。

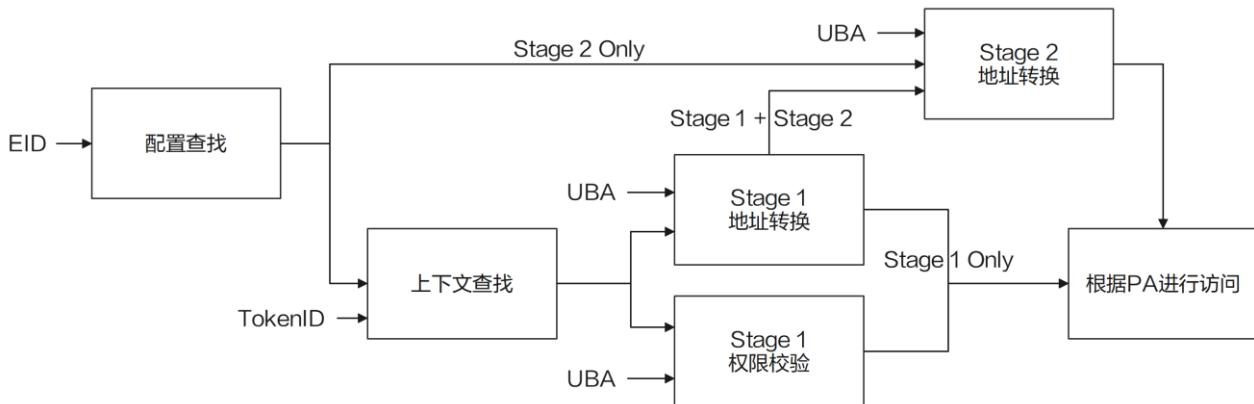


图 9-10 虚拟化场景下 UMMU 整体功能流程示意

两个阶段可单独配置启用情况，如图 9-11 所示。其中 IPA 是 UBA 转换为 PA 的中间物理地址：

1. Stage 1 地址转换：将 UBA 转换为 IPA，该流程相关信息存放在 TCTE 中。
2. Stage 2 地址转换：将 IPA 转换为 PA，该流程相关信息存放在 TECTE 中。

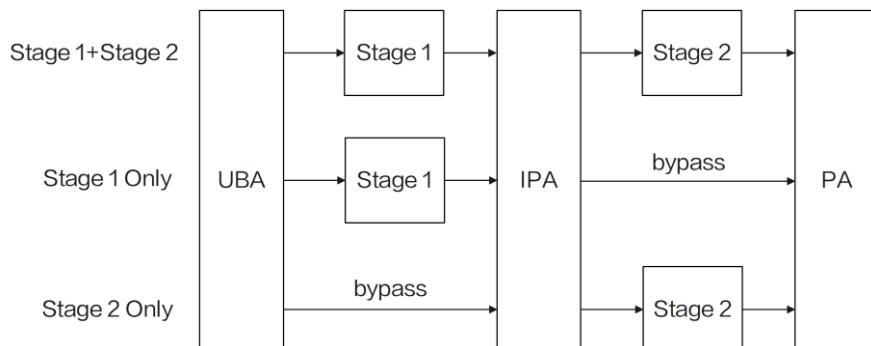


图 9-11 Stage 1 和 Stage 2 地址转换

两阶段地址转换流程如图 9-12 所示，主要步骤为：

1. 通过配置查找，从 TECTE 中获取 Stage 2 MATT 基地址以及 TCT 基地址。如果 Stage 1 和 Stage 2 均启用，UMMU 应对 TCT 基地址进行 Stage 2 地址转换，将 IPA 转换为 PA；如果 Stage 1 Only，则跳过 TCT 的 Stage 2 地址转换；如果 Stage 2 Only，则跳过上下文查找。
2. 通过上下文查找，从 TCTE 中获取 Stage 1 MATT 基地址。
3. 对 UBA 进行 Stage 1 地址转换，可能包括多级查找。
  - (1) 第 x 级查找开始前，需要对 Level x MATT 基地址进行 Stage 2 地址转换。

(2) 第 x 级查找完成后, 获得 Level x MATT 基地址。

4. 完成最后一级 Stage 1 地址转换后, UMMU 可获得 UBA 对应的 IPA; 完成最后一级 Stage 2 地址转换后, UMMU 可获得 UBA 对应的 PA。

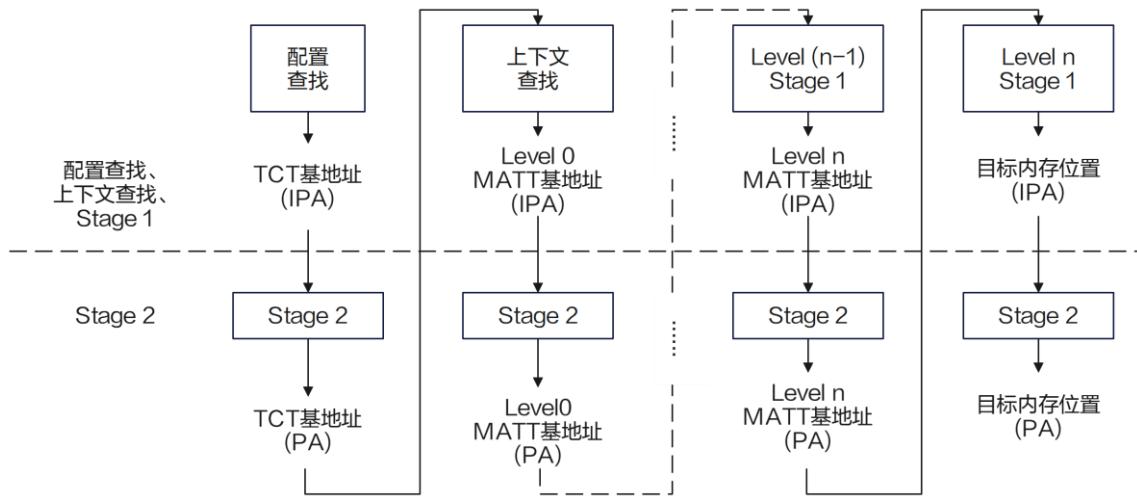


图 9-12 两阶段地址转换流程示意

地址转换具体流程、各级 MATT 的数据结构及相关字段含义参照 UMMU 所在 UBPU 架构规定, 此处不做定义。

## 9.4.4 UMMU 权限校验

### 9.4.4.1 一般要求

UMMU 在进行地址转换时, 可同步执行权限校验流程。经过配置查找和上下文查找后, 若当前被访问的 Entity 启用了权限校验 (见 TECTE.MAPT\_EN), 则 UMMU 可获得权限空间的上下文信息。UMMU 根据 UBA 在 MAPT 中查找对应内存的权限信息, 并与内存访问请求携带的权限进行比对, 最终得出此次内存访问是否通过校验。

MAPT 分为单表项 MAPT 和多级 MAPT, 由 TECTE.MAPT\_MODE 决定。具体存放形式等其它相关信息可以在 TCTE 中获得。

UMMU 的权限表可以提供给非特权级软件直接操作, 用户可以操作的 MAPT 空间被限定在一个 MAPT Base Block 内部, 特权级软件创建 TCT 表项时会分配 MAPT Base Block Address, 可以确保权限表的隔离性。

#### 9.4.4.2 权限查找

##### 9.4.4.2.1 MAPT 存放形式

MAPT 采用一个或多个连续内存块( MAPT Block )进行存放。TCTE 中提供了存放 L0 MAPT 的 MAPT Block ( MAPT Base Block ) 的基地址等信息。L0 MAPT 的基地址即为 MAPT Base Block 的基地址。

若后续各级的 MAPT 均存放于 MAPT Base Block 中，则需要通过偏移进行定位，规则如下：L<sub>x</sub> MAPT 基地址 = TCTE.MAPT\_BBA:5'b0 + L(x-1) MAPTE.Next\_Level\_Offset。

仅使用 MAPT Base Block 存放时，各级 MAPT 索引方式如图 9-13 所示，对应的权限查找如图 9-14 所示。

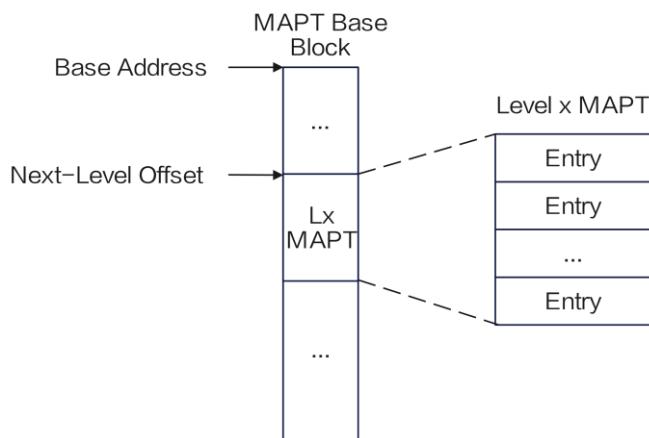


图 9-13 仅使用 MAPT Base Block 时，各级 MAPT 索引方式示意

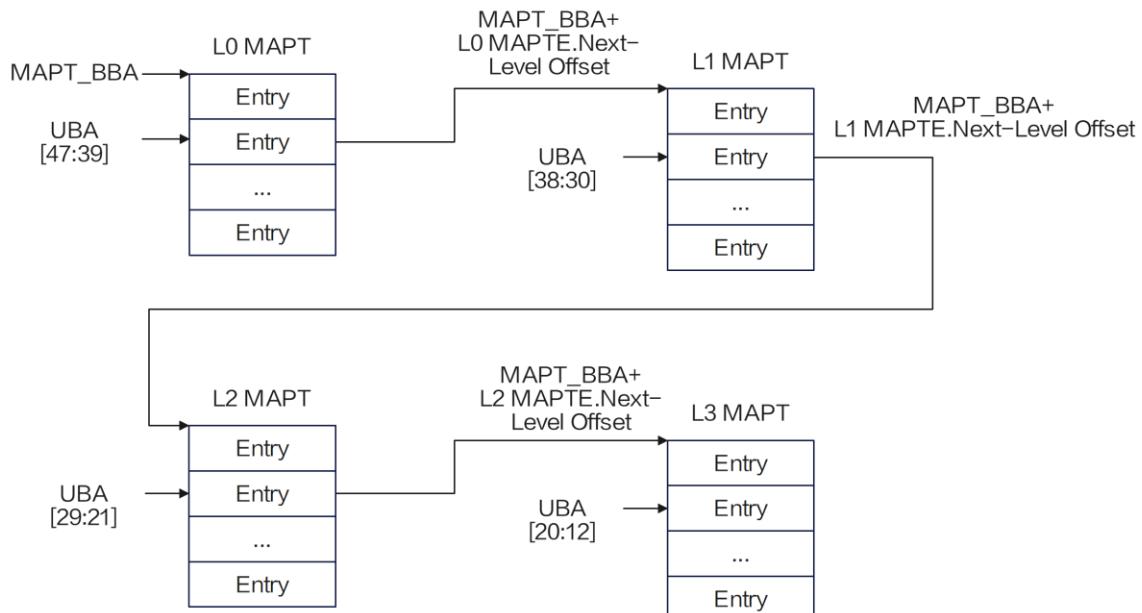


图 9-14 仅使用 MAPT Base Block 时权限查找流程示意

若各级 MAPT 需要多个内存块进行存放，则需要使用 MAPT Block Table 记录所用内存块信息。TCTE 中提供了 MAPT Block Table 的基地址等信息。此时，L0 MAPT 的基地址仍为 MAPT Base Block 的基地址。后续各级的 MAPT 则需要通过所在内存基地址及对应偏移进行定位，流程如下：

- 若 L(x-1) MAPTE.N 取值为 1b'0，则：  
L<sub>x</sub> MAPT 基地址 = L(x-1)\_Block\_BA + L(x-1) MAPTE.Next\_Level\_Offset，其中 L(x-1)\_Block\_BA 为 L(x-1) MAPT 所在内存块基地址。
- 若 L(x-1) MAPTE.N 取值为 1b'1，则：  
L<sub>x</sub> MAPT 基地址 = Next\_Block\_BA + L(x-1) MAPTE.Next\_Level\_Offset，其中 Next\_Block\_BA 为 L<sub>x</sub> MAPT 所在内存块基地址，可以 L(x-1) MAPTE.Next\_Level\_Index 为索引从 MAPT Base Block 中获得。

MAPT Block Table Entry 数据结构格式如图 9-15 所示。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
3~0	MAPT_BA[31:5]																RSVD																V
7~4	MAPT_BS				RSVD								MAPT_BA[47:32]																				

图 9-15 内存访问权限表 Block Table 表项

MAPT Block Table Entry 数据结构格式中各域段含义见表 9-4。

表 9-4 内存访问权限表 Block Table 表项域段描述

域段名称	位宽 ( Bits )	描述
V	1	Valid, 指示当前表项是否有效: 1b'0: 无效; 1b'1: 有效。
MAPT_BA	43	MAPT Block Address, 表示 MAPT Block 基地址中的[47:5]。
MAPT_BS	4	MAPT Block Size, 表示 MAPT Block 大小, 计算方式为 4KB * 2 <sup>MAPT_BS</sup>

用多个 MAPT Block 时，各级 MAPT 索引方式如图 9-16 所示，对应的权限查找如图 9-17 所示。

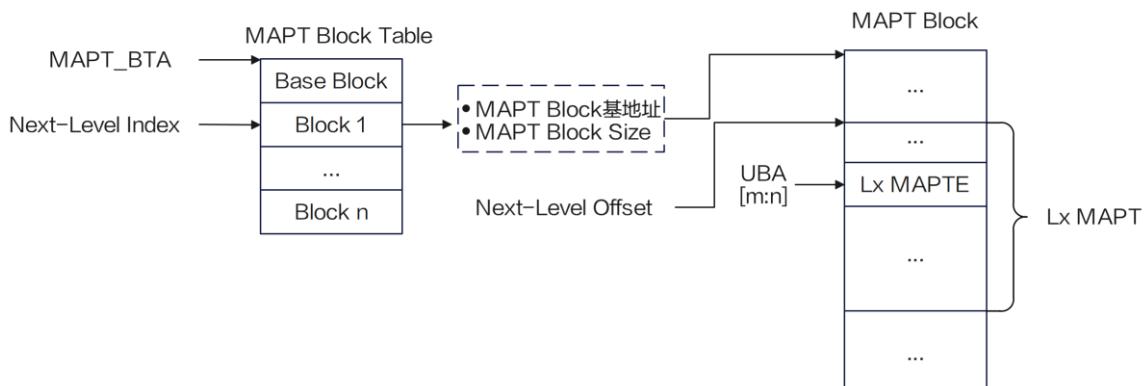


图 9-16 使用多个 MAPT Block 时，各级 MAPT 索引方式示意

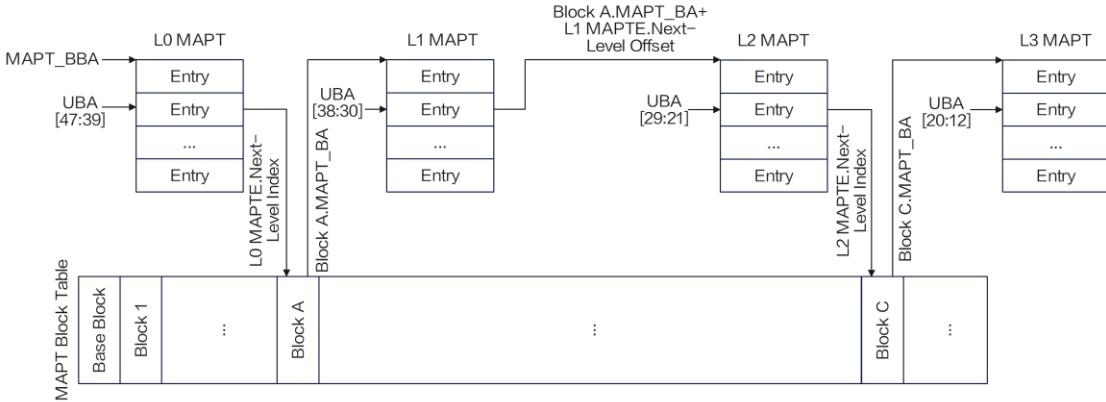


图 9-17 使用多个 MAPT Block 时权限查找流程示意

MAPT 可由非特权级软件直接管理。为确保权限表的隔离性，MAPT Block 的物理内存应由特权级软件分配，与非特权级软件的虚拟地址建立映射。该段内存不会被换出，非特权级软件对 MAPT 进行访问不会触发缺页异常。此外，应确保非特权级软件对 MAPT Block 的访问不超过 MAPT Block 大小。

#### 9.4.4.2.2 单表项 MAPT

对于单表项 MAPT，UMMU 可直接通过 TCTE.MAPT\_BBA 获得唯一表项 Unique Memory Address Permission Table Entry (UMAPTE)。该表项中包含了一个 UBA 范围 (通过 Base 和 Limit 域段表示)。UMMU 应判断内存访问所携带的 UBA 是否落入该范围，以决定继续进行其它权限信息的校验或终止当前的内存访问 (见节 9.4.4.3.2)。单表项 MAPT 下的权限查找流程如图 9-18 所示。

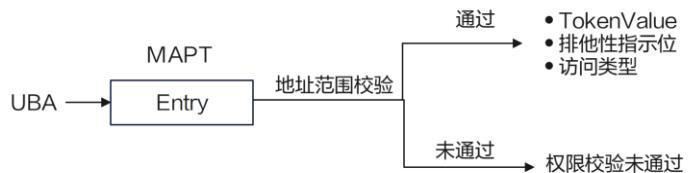


图 9-18 单表项 MAPT 权限查找流程示意

UMAPTE 数据结构格式如图 9-19 所示。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
3~0																																
7~4																																
11~8																																
15~12	TOKEN_CHK	IMPL_DEF																														
19~16																																
23~20	IMPL_DEF																															
27~24																																
31~28																																

图 9-19 唯一内存访问权限表表项

UMAPTE 数据结构格式中各字段含义见表 9-5。

**表 9-5 唯一内存访问权限表表项字段描述**

域段名称	位宽( Bits )	描述
V	1	Valid, 指示当前 UMAPTE 是否有效。 1'b0: UMAPTE 无效; 1'b1: UMAPTE 有效。 当 UMAPTE.V==1'b0 时, 本次内存访问将被终止, 向 User 返回错误信息, 并上报事件。
E_Bit	1	Exclusive-Bit, 校验规则见节 9.4.4.3.3。 当 E_Bit 校验失败时, 需上报事件。
Permission	10	指示当前 UMAPTE 指示的地址范围支持的访问类型。该字段各个位对应的类型如下: Bit 0: 写 Bit 1: 读 Bit 2: 原子操作 Bits [9:3]: 保留 单个访问类型对应位取值含义如下: 1'b0: 不支持该类型访问; 1'b1: 支持该类型访问;
Base	48	表示该 UMAPTE 对应地址范围的起始地址。
TOKEN_CHK	1	Token Check, 指示是否需要做访问凭据校验: 1'b0: 不需要做访问凭据校验; 1'b1: 需要做访问凭据校验。
Limit	48	表示该 UMAPTE 对应地址范围的地址上限。
TokenVal0	32	用于校验的主 TokenValue。
TokenVal1	32	用于校验的备 TokenValue。

#### 9.4.4.2.3 多级 MAPT

多级 MAPT 支持 4KB 和 2MB 两种粒度 (由 TCTE.RTGS 决定)。4KB 粒度下, 多级 MAPT 最多支持 4 级 MAPT; 2MB 粒度下, 多级 MAPT 最多支持 3 级 MAPT。UMMU 应以 UBA 为索引逐级进行权限查找, 各级表项均包含一个 UBA 范围。通过比对该范围以及内存访问所携带的 UBA, UMMU 将决定进行其它权限信息校验、继续对下一级 MAPT 进行查找或终止当前的内存访问。

采用多级 MAPT 时，各级 MAPT 存放于一个或多个连续内存块中。Level 0 Memory Address Permission Table ( L0 MAPT ) 基地址可在 TCTE 中获得。其余各级 MAPT 的基地址由上一级 MAPTE 中提供的内存块信息以及当前 MAPT 在该内存块中的偏移计算获得。

以 4KB 粒度为例，权限查找流程如图 9-20 所示，具体如下：

1. 从 TCTE 中获得 L0 MAPT 基地址（见 TCTE.MAPTE\_BBA），以 UBA[47:39]为索引，获得对应的 L0 MAPTE，执行以下步骤：
  - (1) 若 UBA 落入当前 L0 MAPTE 的 UBA 范围，则 UMMU 将获取表项中其它权限信息，继续进行校验。
  - (2) 若 UBA 未落入 L0 MAPTE 的 UBA 范围，且当前 L0 MAPTE 包含 L1 MAPT 基地址相关信息（由 L0 MAPTE.T 决定），则 UMMU 将对 L1 MAPT 进行查找。
  - (3) 若 UBA 未落入 L0 MAPTE 的 UBA 范围，且当前 L0 MAPTE 不包含 L1 MAPT 基地址相关信息（由 L0 MAPTE.T 决定），则此次内存访问未通过权限校验，UMMU 将终止当前的内存访问。
2. 根据 L0 MAPTE 提供的信息获得 L1 MAPT 基地址，以 UBA[38:30]为索引，获得对应的 L1 MAPTE，执行以下步骤：
  - (1) 若 UBA 落入当前 L1 MAPTE 的 UBA 范围，则 UMMU 将获取表项中其它权限信息，继续进行校验。
  - (2) 若 UBA 未落入 L1 MAPTE 的 UBA 范围，且当前 L1 MAPTE 包含 L2 MAPT 基地址相关信息（由 L1 MAPTE.T 决定），则 UMMU 将对 L2 MAPT 进行查找。
  - (3) 若 UBA 未落入 L1 MAPTE 的 UBA 范围，且当前 L1 MAPTE 不包含 L2 MAPT 基地址相关信息（由 L1 MAPTE.T 决定），则此次内存访问未通过权限校验，UMMU 将终止当前的内存访问。
3. 根据 L1 MAPTE 提供的信息获得 L2 MAPT 基地址，以 UBA[29:21]为索引，获得对应的 L2 MAPTE，执行以下步骤：
  - (1) 若 UBA 落入当前 L2 MAPTE 的 UBA 范围，则 UMMU 将获取表项中其它权限信息，继续进行校验。
  - (2) 若 UBA 未落入 L2 MAPTE 的 UBA 范围，且当前 L2 MAPTE 包含 L3 MAPT 基地址相关信息（由 L2 MAPTE.T 决定），则 UMMU 将对 L3 MAPT 进行查找。
  - (3) 若 UBA 未落入 L2 MAPTE 的 UBA 范围，且当前 L2 MAPTE 不包含 L3 MAPT 基地址相关信息（由 L2 MAPTE.T 决定），则此次内存访问未通过权限校验，UMMU 将终止当前的内存访问。
4. 根据 L2 MAPTE 提供的信息获得 L3 MAPT 基地址，以 UBA[20:12]为索引，获得对应的 L3 MAPTE，执行以下步骤：
  - (1) 若 UBA 落入当前 L3 MAPTE 的 UBA 范围，则 UMMU 将获取表项中其它权限信息，继续进行校验。

- (2) 若 UBA 未落入 L3 MAPTE 的 UBA 范围，则此次内存访问未通过权限校验，UMMU 将终止当前的内存访问。

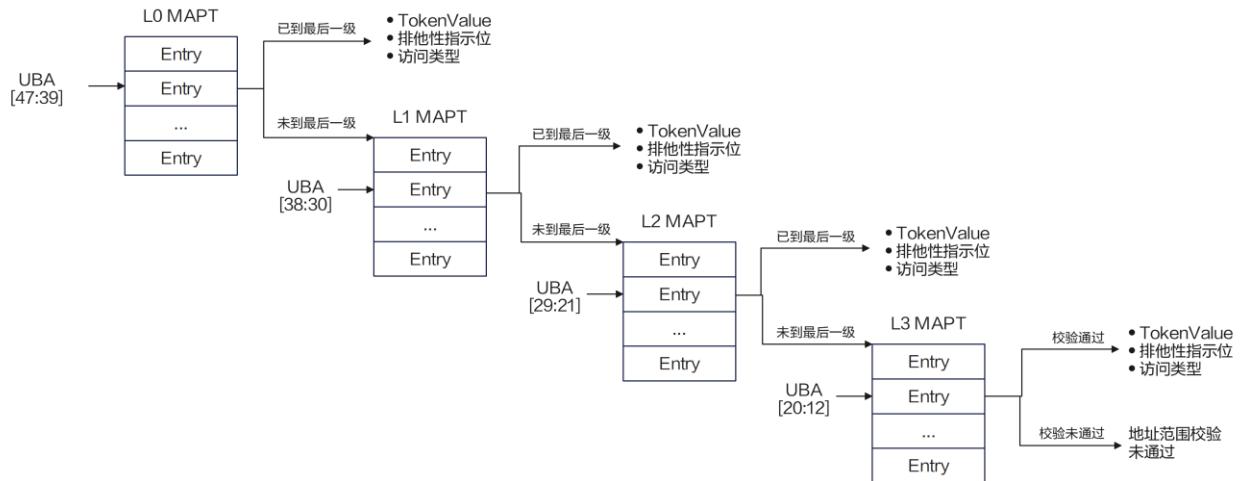


图 9-20 4KB 粒度下的多级 MAPT 权限查找流程示意

各级 MAPTE 的数据结构格式如图 9-21 到图 9-25 所示。其中，4KB 粒度和 2MB 粒度的 L2 MAPTE 格式相同。

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
3~0	Next_Level_Offset[19:0]										RSVD	Permission										E	I	N	T	V																	
7~4	Next_Level_Index[15:0]										RSVD	Next_Level_Offset[29:20]																															
11~8	Base[31:0]															Base[38:32]																											
15~12	TOKEN_CHK	IMPL_DEF	RSVD										Base[38:32]																														
19~16	Limit[31:0]															Limit[38:32]																											
23~20	IMPL_DEF	RSVD										Limit[38:32]																															
27~24	TokenVal0															TokenVal1																											
31~28	TokenVal1																																										

图 9-21 Level 0 内存访问权限表表项

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0												
3~0	Next_Level_Offset[19:0]										RSVD	Permission										E	I	N	T	V																		
7~4	Next_Level_Index[15:0]										RSVD	Next_Level_Offset[29:20]																																
11~8	RSVD	Base[29:0]															RSVD																											
15~12	TOKEN_CHK	IMPL_DEF	RSVD										RSVD																															
19~16	RSVD	Limit[29:0]										RSVD																																
23~20	IMPL_DEF	RSVD										RSVD																																
27~24	TokenVal0															TokenVal1																												
31~28	TokenVal1																																											

图 9-22 Level 1 内存访问权限表表项

## 9 内存管理

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																	
3~0	Next_Level_Offset[19:0]												RSVD	Permission				E_Bit	N	T	V																																																																												
7~4	Next_Level_Index[15:0]												RSVD	Next_Level_Offset[29:20]																																																																																			
11~8	RSVD												Base[20:0]																																																																																				
15~12	TOKEN_CHK	IMPL_DEF	RSVD																																																																																														
19~16	RSVD												Limit[20:0]																																																																																				
23~20	IMPL_DEF			RSVD																																																																																													
27~24	TokenVal0																																																																																																
31~28	TokenVal1																																																																																																

图 9-23 4KB 粒度下的 Level 2 内存访问权限表表项

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																				
3~0	RSVD												Permission				E_Bit	RSVD	T	V																																																																
7~4	RSVD																																																																																			
11~8	RSVD												Base[20:0]																																																																							
15~12	TOKEN_CHK	IMPL_DEF	RSVD																																																																																	
19~16	RSVD												Limit[20:0]																																																																							
23~20	IMPL_DEF			RSVD																																																																																
27~24	TokenVal0																																																																																			
31~28	TokenVal1																																																																																			

图 9-24 2MB 粒度下的 Level 2 内存访问权限表表项

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																						
3~0	RSVD												Permission				E_Bit	RSVD	T	V																																																																		
7~4	RSVD																																																																																					
11~8	RSVD												Base[11:0]																																																																									
15~12	TOKEN_CHK	IMPL_DEF	RSVD																																																																																			
19~16	RSVD												Limit[11:0]																																																																									
23~20	IMPL_DEF			RSVD																																																																																		
27~24	TokenVal0																																																																																					
31~28	TokenVal1																																																																																					

图 9-25 Level 3 内存访问权限表表项

MAPTE 数据结构格式中各域段含义见表 9-6。

表 9-6 多级 MAPT 下各级 MAPTE 域段描述

域段名称	位宽( Bits )	描述
V	1	Valid, 指示当前 MAPTE 是否有效。 1'b0: MAPTE 无效; 1'b1: MAPTE 有效。

域段名称	位宽( Bits )	描述
		若内存访问命中的 MAPTE.V==1'b0，则本次内存访问将被终止，向 User 返回错误信息，并上报事件。
T	1	Terminate, 指示当前 MAPTE 是否包含下一级 MAPT 基地址相关信息。 1'b0: 包含下一级 MAPT 基地址相关信息； 1'b1: 不包含下一级 MAPT 基地址相关信息。
N	1	Not same block, 指示下一级 MAPT 是否在所存放的 MAPT Block 是否与当前 MAPTE 相同。 1'b0: 下一级 MAPT 所在 MAPT Block 与当前 MAPTE 相同。 1'b1: 下一级 MAPT 所在 MAPT Block 与当前 MAPTE 不同。
E_Bit	1	Exclusive-Bit, 校验规则见节 9.4.4.3.3。 当 E_Bit 校验失败时，需上报事件。
Permission	10	指示当前 MAPTE 指示的地址范围支持的访问类型。该域段各个位对应的类型如下： Bit 0: 写 Bit 1: 读 Bit 2: 原子操作 Bits [9:3]: 保留 单个访问类型对应位取值含义如下： 1'b0: 不支持该类型访问； 1'b1: 支持该类型访问；
Next_Level_Offset	30	表示下一级 MAPT 相对所在 MAPT Block 基地址中的偏移。
Next_Level_Index	16	表示下一级 MAPT 所在 MAPT Block 在 Block Table 中的索引。
TOKEN_CHK	1	Token Check, 指示是否需要做访问凭据校验： 1'b0: 不需要做访问凭据校验； 1'b1: 需要做访问凭据校验。
Base	48	Range table 权限地址范围的 Base 地址信息 ( 4KB 粒度 )： Level 0 查表时, base[38:0]有效； Level 1 查表时, base[29:0]有效； Level 2 查表时, base[20:0]有效； Level 3 查表时, base[11:0]有效； Range table 权限地址范围的 Base 地址信息 ( 2MB 粒度 )： Level 0 查表时, base[38:0]有效；

域段名称	位宽( Bits )	描述
		Level 1 查表时, base[29:0]有效; Level 2 查表时, base[20:0]有效; 无 Level 3。
Limit	48	Range table 权限地址范围的 Limit 地址信息 ( 4KB 粒度 ) : Level 0 查表时, limit[38:0]有效; Level 1 查表时, limit[29:0]有效; Level 2 查表时, limit[20:0]有效; Level 3 查表时, limit[11:0]有效; Range table 权限地址范围的 Limit 地址信息 ( 2MB 粒度 ) : Level 0 查表时, limit[38:0]有效; Level 1 查表时, limit[29:0]有效; Level 2 查表时, limit[20:0]有效; 无 Level 3。
TokenVal0	32	用于校验的主 TokenValue。
TokenVal1	32	用于校验的备 TokenValue。

#### 9.4.4.3 权限比对

##### 9.4.4.3.1 一般要求

对于由 UMMU 处理的内存访问，权限比对具体内容包括：

- 访问凭据
- 排他性指示位
- 访问类型

除了上述内容外，UMMU 还应结合地址转换流程随附的权限校验，得出此次内存访问的权限校验结果。在本小节中，如未特殊说明，MAPTE 默认指代 UMMU 根据当前处理的内存访问请求进行权限查找后获得的表项。

##### 9.4.4.3.2 访问凭据校验

访问凭据校验的启用情况由 MAPTE.TOKEN\_CHK 决定。MAPTE 中包含主 TokenValue 和备 TokenValue。当内存访问所携带的 TokenValue 与其中任意一个 TokenValue 相等时，则此次访问通过访问凭据校验；否则此次访问未通过权限校验（如图 9-26 所示）。



图 9-26 TokenValue 校验

注：在一般场景下，当 MAPTE 中的 TokenValue 发生修改时，需要由软件对 UMMU 下发无效化命令，避免过期 Permission Lookaside Buffer( PLB )表项影响权限校验流程的正确性。Positive PLB 特性要求 UMMU 在基于 PLB 表项进行 TokenValue 校验不通过时，追加权限查找流程以保证正确性。这一特性允许 MAPTE 中的 TokenValue 发生修改时，软件无需下发无效化命令。相关流程的伪代码如下所示：

```

if (PLB 命中){
    得到 TokenValue;
    if (TokenValue 匹配)
        通过 TokenValue 校验 ;
    else{
        无效对应 PLB;
        通过权限查找获取 MAPTE;
        if (TokenValue 匹配)
            通过 TokenValue 校验 ;
        else
            权限校验未通过 ;
    }
}
else{
    通过权限查找获取 MAPTE;
    if (TokenValue 匹配)
        通过 TokenValue 校验 ;
    else
        权限校验未通过 ;
}

```

#### 9.4.4.3.3 排他性校验

被访问位置的排他性指示位 ( E\_Bit ) 存放于 TCTE 和 MAPTE 中，分别在上下文查找阶段和权限校验阶段时与此次内存访问携带的 E\_Bit ( 记作 Input.E\_Bit ) 进行比对。

UMMU 经过上下文查找获得 TCTE 后，应根据以下规则对 Input.E\_Bit 进行校验 ( 如表 9-7 所示 )：

- 若 Input.E\_Bit 取值为 1'b1，或 Input.E\_Bit 与 TCTE.E\_Bit 取值均为 1'b0，则此次访问通过当前校验，UMMU 可继续执行后续的地址转换和权限校验；
- 若 Input.E\_Bit 取值为 1'b0 且 TCTE.E\_Bit 取值为 1'b1，则此次访问未通过权限校验，需上报事件。

**表 9-7 TCTE 排他性校验规则**

	TCTE.E_Bit==1'b0	TCTE.E_Bit ==1'b1
Input.E_Bit==1'b0	pass	deny
Input.E_Bit==1'b1	pass	pass

UMMU 经过权限查找获得 MAPTE 后，应根据以下规则对 Input.E\_Bit 进行校验（如表 9-8 所示）：

- 若 Input.E\_Bit 取值为 1'b1，或 Input.E\_Bit 与 MAPTE.E\_Bit 取值均为 1'b0，则此次访问通过排他性校验；
- 若 Input.E\_Bit 取值为 1'b0 且 MAPTE.E\_Bit 取值为 1'b1，则此次访问未通过权限校验，需上报事件。

**表 9-8 MAPTE 排他性校验规则**

	MAPTE.E_Bit==1'b0	MAPTE.E_Bit==1'b1
Input.E_Bit==1'b0	pass	deny
Input.E_Bit==1'b1	pass	pass

#### 9.4.4.3.4 访问类型校验

访问类型包括读、写、原子操作三种类型。被访问内存的访问类型权限信息由 MAPTE.Permission 域段提供。若当前内存访问的类型在该域段中的对应位启用，则校验通过；否则此次访问未通过权限校验。

#### 9.4.4.3.5 结合地址转换的权限校验结果

UMMU 处理地址转换会通过 MATT 获得地址转换随附的访问权限，在此基础上叠加权限校验的信息。

综合两方面结果，权限校验的最终结果如下：

- 当与 MAPTE 相关的权限校验未通过时，内存访问终止，并上报权限校验相关事件。
- 当地址转换随附的权限校验未通过时，内存访问终止，并上报地址转换相关事件。
- 当两方面的校验均通过时，本次内存访问通过权限校验。

## 9.5 UB Decoder 功能与流程

### 9.5.1 PA-UBMD 转换流程

PA 是 UB Decoder 的输入，经过 UB Decoder 后可获得标识远端内存的 UBMD。

UB Decoder 通过查找 UB Decoder Page Table (本小节中简称为 Page Table) 来完成 PA-UBMD 转换。Page Table 包括两级数据结构：Level 0 Page Table (L0 Page Table) 和 Level 1 Page Table (L1 Page Table)。

L0 Page Table 中可能包含的表项类型有：

- L0 Page Table Entry ( L0 PTE )。
- L0 Page Table Range Entry ( L0 PTRE )。

本章节以 44 bits 位宽 PA、8B 大小 L0 PTE 和 64B 大小 L0 PTRE 为例描述查表过程。当 UB Decoder 对 L0 Page Table 进行查找时，使用 PA[43:35]作为索引，获取一个大小为 64B 的大块表项( Bulk Entry )，随后根据 Bulk Entry 的 Type 域段判断其实际类型，分别执行相应操作：

- 若该 Bulk Entry 由 8 个 L0 PTE 组成，需要进一步使用 PA[34:32]作为索引从中选取 1 个 L0 PTE ( 如图 9-27 所示 )。
- 该 Bulk Entry 是一个 L0 PTRE ( 如图 9-28 所示 )，则需要使用 L0 PTRE 中包含的地址范围与 PA 进行比较：若  $\text{Mem\_Base} \leq \text{PA}[34:20] \leq \text{Mem\_Limit}$ ，那么说明 PA 落入该 L0 PTRE 对应的地址范围。

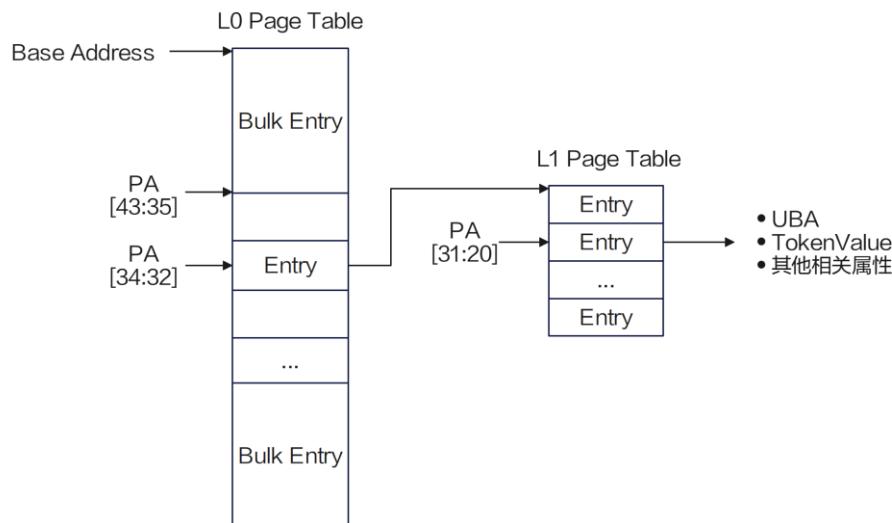


图 9-27 L0 Page Table 查找结果为 L0 PTE 时，UB Decoder 整体转换流程

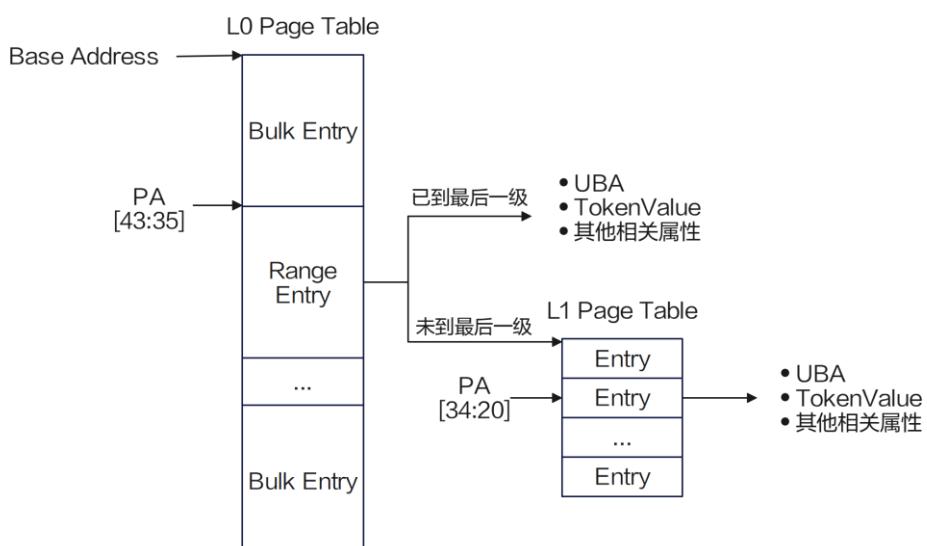


图 9-28 L0 Page Table 查找结果为 L0 PTRE 时，UB Decoder 转换流程

L0 PTE 和 L0 PTRE 均包含 L1 Page Table 的基地址。对于以下场景，UB Decoder 需要继续查找 L1 Page Table。

- L0 Page Table 查找结果为 L0 PTE 时，需要使用 PA[31:20]查找 L1 Page Table。
- L0 Page Table 查找结果为 L0 PTRE 但未落入对应地址范围时，需要使用 PA[34:20]查找 L1 Page Table。

### 9.5.2 UBA 计算

以 PA 为索引所命中的 L0 PTRE 或 L1 PTE 中，均包含了发起内存访问所需的信息，其中 UBA 需要结合 UBA\_BASE 域段和 PA 计算得出。其计算规则如下：

$$\text{UBA} = \text{UBA\_BASE}:12'b0 + 29'b0:\text{PA}[34:0]$$

# 10 资源管理

## 10.1 概述

UBFM 是 UB Domain 的管理者，负责 Domain 内的互连、通信和计算资源管理，动态处理系统运行过程中产生的事件。UB Domain 内所有 Entity 和 Port 资源都是 UBFM 的管理对象，UBFM 基于 Entity 的全局唯一身份标识（Globally Unique Identifier, GUID）识别和调度资源，并动态管理 Entity 的通信对象身份标识（EID），使不同 Entity 之间可基于对方的 EID 实现直接访问。

一个 Entity 在访问另一个 Entity 所在 UBPU 的内存时，携带该 UBPU 发放的 Token 作为访问凭据，Target Entity 基于 Token 对事务请求进行鉴权。在安全管控下，一个 Entity 在获得 Exclusive 权限后可仅携带 TokenID 作为访问凭据去访问目标内存，包括 UBPU 内存和其它 Entity 的资源空间，从而提升访问效率。Entity 基于 UB 分区实现访问隔离，UBFM 将 Entity 加入指定的分区，属于不同分区的 Entity 访问隔离。

所有 Entity 声明的功能或服务在 UB Domain 内都可以是共享的资源，在 UBFM 完成资源调度和注册后，一个 UBPU 可以基于其 UB Controller 提供的接口调用其它 Entity 声明的功能或服务。UB 可支持多种远端功能或服务的调用方式，包括 MMIO 映射方式、消息通信方式以及 URPC 调用方式等，UBPU 可根据具体的系统环境选择合适的调用方式。

UB 定义了一套错误处理和事件通知机制，一个设备在运行过程中产生的本地事件，可以使用 UB 信号中断或消息通知方式通知对应的故障处理单元进行处理。例如，设备运行过程中产生的错误可以通过 Error Message 上报给 UBFM 处理，Entity 在处理功能调用时产生的错误可通过中断通知对应的 UBPU 处理，Target Entity 在处理内存访问请求时产生的异常可通过响应包携带的错误码或 Poison 标记通知 Initiator Entity 处理。

## 10.2 基本概念

### 10.2.1 协议组件

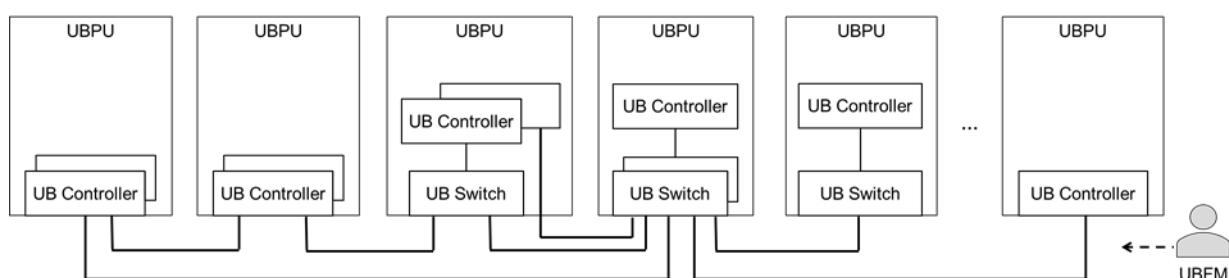


图 10-1 UB Domain 示意图

如上图所示，一个 UB Domain 的范围可以对应于物理世界中的一个超节点或一台服务器，包含若干个支持不同功能或服务的 UBPU。在 UB Domain 内，支持使用任意拓扑连接 UBPU，允许系统资源进行任意组合。

UB Controller 和 UB Switch 是实现 UB 协议栈的不同组件，分别位于系统中不同的拓扑位置，UB Controller 位于叶子节点位置。UB Switch 提供路由服务，负责互连 UB Controller，可通过级联方式连接更多的 UB Controller。UB Controller 可为本 UBPU 提供远端内存访问和消息通信等服务，并负责声明本 UBPU 可对外提供的功能或服务，使其可被其它 UBPU 调用；UBPU 可通过其本地 UB Controller 提供的接口调用其它 UBPU 对外提供的功能或服务。

一个 UBPU 内可具有多个 UB Controller，也可具有多个 UB Switch，在具体实现中，UBPU 可按需包含不同数量的 UB Controller 和 UB Switch，可通过 UB Controller 或 UB Switch 的端口连接其它 UBPU，本规范不做约束。

UBFM 是 UB Domain 的管理者，可通过管理命令发现并管理本 Domain 内的资源，并动态处理系统运行过程中产生的事件。

1. 一个大型 UB Domain 中可部署多个 UBFM 实例，一个 UBFM 实例可只负责 UB Domain 部分范围的管理，该部分范围可称之为 Sub Domain；多个 UBFM 实例应协同承担 UBFM 职责，具体的协同实现方式不在本规范定义。
2. UBFM 实例的部署应保证可信和可靠性，具体的实现方式不在本规范定义。
3. 在服务器内，UBFM 职责可由主机系统软件承担。

UB Controller 应按需支持如下能力：

1. 接受 UBFM 的管理，响应管理命令；
2. 同步或异步的远端内存访问、消息通信等服务；
3. 网络层路由机制；
4. 资源池化，如内存资源池、Entity 资源池等；
5. 访问隔离；
6. 复位；
7. 错误处理；
8. 中断；
9. 虚拟化；
10. 其它厂商自定义功能；

UB Switch 应按需支持如下能力：

1. 接受 UBFM 的管理，响应管理命令；
2. 网络层路由机制；
3. 访问隔离；
4. 复位；
5. 错误处理；

6. 上报中断；
7. 其它厂商自定义功能；

### 10.2.2 配置管理模型

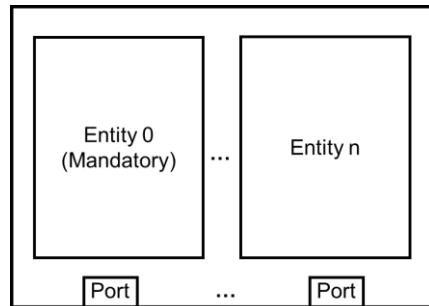


图 10-2 配置管理模型结构示意图

UB Controller 和 UB Switch 采用相同的模型结构。如上图所示，UB Controller 和 UB Switch 均由 Entity 和 Port 组成，应包含至少一个 Entity 以及至少一个 Port。Port 可以是 physical Port 或 virtual Port ( vPort )，由一个模型结构内的所有 Entity 共享使用。

1. 一个模型结构包含的多个 Entity 应从 0 开始递增编号，且必须连续编号。若包含了  $N \leq 65536$  个 Entity，则 Entity 编号为 Entity 0, Entity 1, … , Entity N-1，应至少包含 Entity 0。
2. 一个模型结构包含的多个 Port 应从 0 开始递增编号，且必须连续编号。若包含了  $M \leq 16384$  个 Port，则 Port 编号为 Port 0, Port 1, … , Port M-1，应至少包含 Port 0。

Entity 是 UB Controller 和 UB Switch 分配其自身资源的基本单元，具有配置空间和资源空间，用于存储配置信息和设备信息。Entity 0 额外承担网络层、数据链路层、物理层等公共资源和其它 Entity 的配置管理功能。Entity 的配置空间与资源空间整体视图如下图所示：

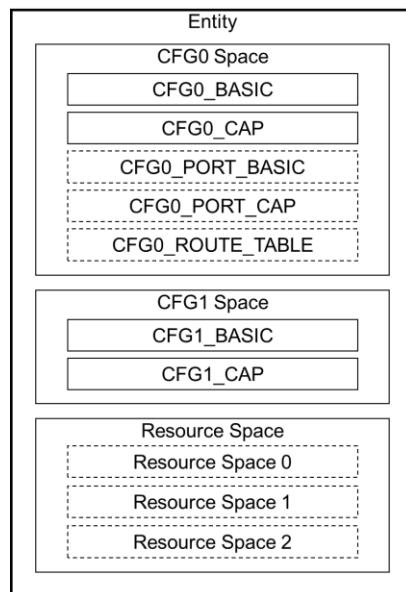


图 10-3 Entity 配置空间与资源空间

1. 每个 Entity 都有 CFG0 和 CFG1 空间，仅 Entity 0 配置空间中存在 CFG0\_PORT\_BASIC、CFG0\_PORT\_CAP 和 CFG0\_ROUTE\_TABLE。
2. 每个 Entity 的资源空间（如有）最多可分成三段。

在生产阶段，每个 Entity 都应被赋予一个 GUID，一个模型结构中的多个 Entity 可以声明不同的功能或服务，使用 Class Code 编码描述。每个 Entity 在被使用前都应被分配一个 EID，用于标识该 Entity 的通信对象身份。

### 10.2.3 资源管理模型

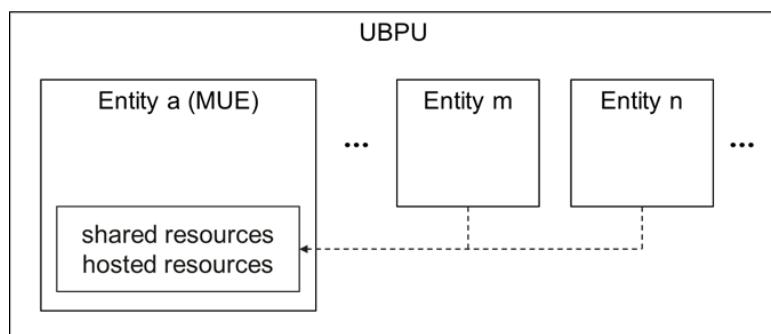


图 10-4 资源管理模型结构示意图

如上图所示，从通信资源管理维度，UBPU 应指定从 Entity 0 开始连续的若干个 Entity 承担 Management UB Entity ( MUE ) 功能，具体数量由实现决定。

MUE 为其所属 UBPU 内的其它 Entity 提供 TP Channel 等共享资源，并负责提供共享资源的管理接口；UBPU 内的其它 Entity 可以使用由对应的 MUE 提供的共享资源，自身不具有对共享资源的管理权限。在虚拟化场景，Entity 可直通给虚机使用，MUE 可用于向对应的多个 Entity 提供 Jetty 上下文等资源的资源托管服务，从而减少虚机的攻击面。

## 10.3 工作机制

### 10.3.1 Entity 访问



图 10-5 EID 示意图

EID 是一个 128-bit ID，有且仅有一个 ID 地址空间。系统管理员应为 UB Domain 规划 EID 前缀，具体规划方式由系统管理员决定，本规范不做约束。一个 UB Domain 可占用一个前缀标识的子 ID 地址空

间,由UBFM规划并可信配置Domain内Entity的EID,其中20-bit EID的0值由协议保留使用,不应被分配。在具体实现中,UB链路上传输的数据包可以携带全长128-bit EID格式,也可以携带简短EID格式(20-bit EID)或不显式地携带EID。

一个Entity可使用一个或多个网络地址实现多端口聚合通信,共享物理端口的多个Entity可使用相同的网络地址。一个Entity被分配EID后,应与其所在拓扑位置的网络地址绑定,该Entity迁移后,应被重新绑定到对应拓扑位置的网络地址。EID与网络地址绑定的具体实现不在本规范定义。

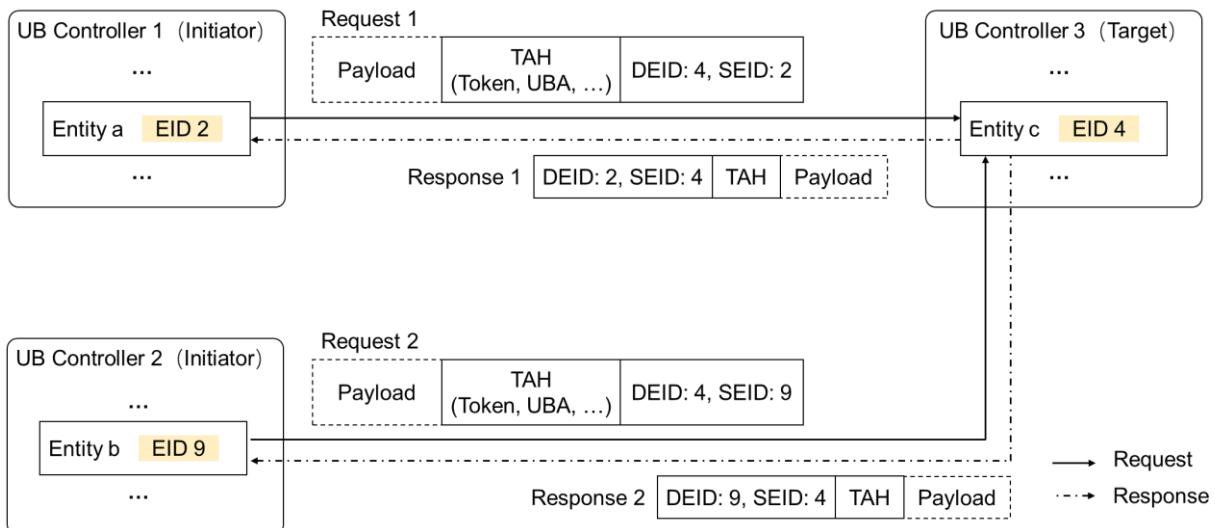


图 10-6 Entity 间访问示意图

如上图所示,一个Entity在访问另一个Entity前,应先获取到目标Entity的EID和目标内存的Token、UBA等信息,该信息可由上层应用指定,或使用其它方式得到,不在本规范定义范围。以内存访问为例,Initiator Entity发起的访问请求包应携带Target Entity的EID(作为DEID)、Token和UBA,以及自身EID(作为SEID)。Target在接收到访问请求包后,使用DEID确认应处理该访问请求的Entity,Entity基于数据包携带的Token进行鉴权防止非法访问。只要持有Target发放的访问凭据,任意Entity可以使用该访问凭据访问Target目标内存,访问凭据可以是Target直接授予的,也可以是其它Entity转授的。

### 10.3.2 Entity 隔离

UB Partition是一组Entity的集合,一个Entity属于且仅属于一个UB Partition,属于不同UB Partition的Entity之间无法互相访问。UB Partition Identifier(UPI)是UB Partition的分区标识,应遵守如下规则:

1. UPI应支持可信配置和防仿冒。
2. Entity的UPI缺省值为0。
3. UPI值为0的Entity,不能与其它Entity通信。
4. 发送端发送数据包时添加UPI,接收端接收数据包时检查UPI,若UPI不匹配,则丢弃数据包。
5. UPI支持15-bit和24-bit两种长度,通信流量可按需使用,详见B.3节。

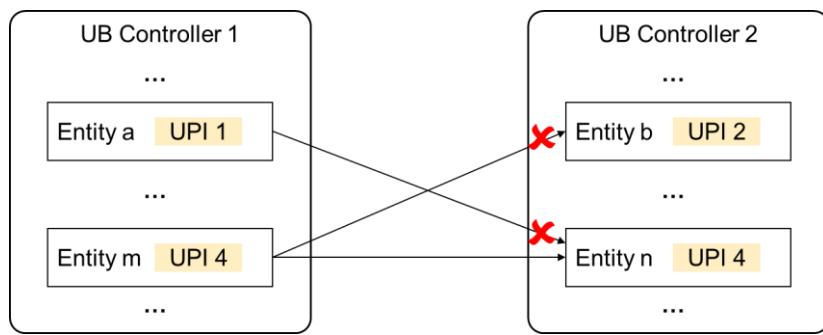


图 10-7 Entity 间访问隔离示意图

如上图所示，Entity m 和 Entity n 属于同一个 UB Partition 4，Entity n 允许 Entity m 访问，不允许 Entity a 访问；Entity b 属于另一个 UB Partition 2，不允许 Entity m 访问。

### 10.3.3 功能调用

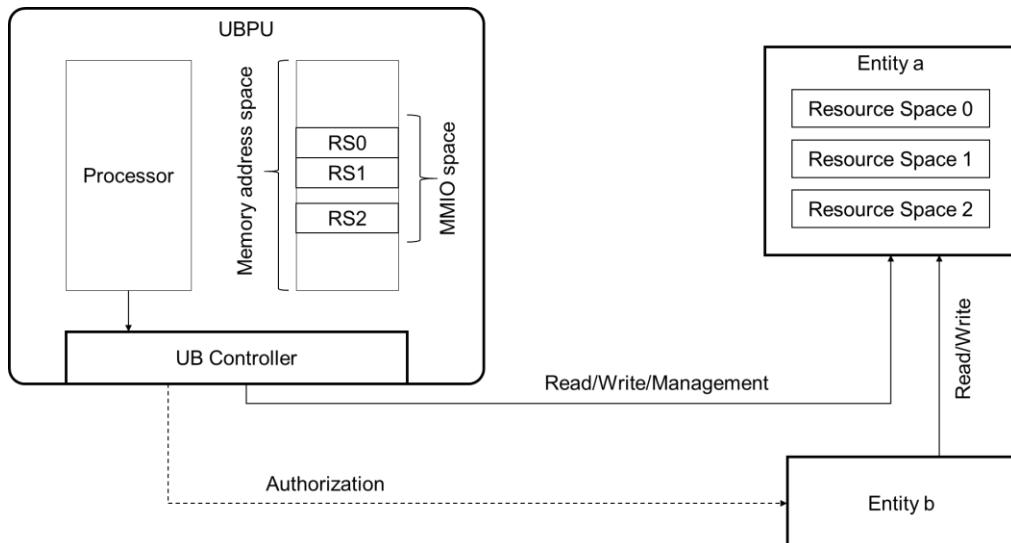


图 10-8 基于 MMIO 的功能调用示意图

如上图所示，Entity a 注册给 UBPU 使用时，UBPU 拥有 Entity a 的部分管理权限。在 UBPU 将 Entity a 的资源空间映射到其自身内存地址空间的 MMIO 空间后，处理器可以通过读写 MMIO 空间的方式实现对 Entity a 声明的功能或服务的调用。当 UBPU 将访问凭据授权给 Entity b 后，Entity b 可以通过 DMA 等方式读写 Entity a 资源空间实现对 Entity a 声明的功能或服务的调用。

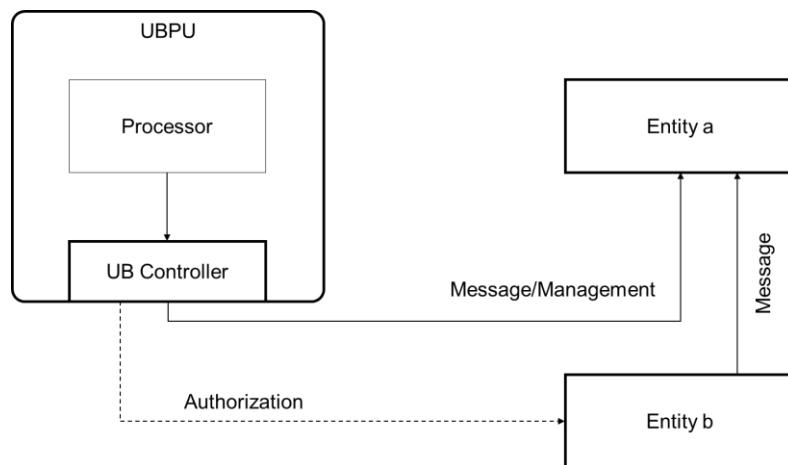


图 10-9 基于 Message 的功能调用示意图

如上图所示，Entity a 被 UBPU 使用时，UBPU 拥有 Entity a 的部分管理权限。UBPU 可使用其本地 UB Controller 提供的消息通信接口发送命令给 Entity a，Entity a 解析并执行该命令，从而使 UBPU 能够调用 Entity a 所声明的功能或服务。当 UB Controller 支持 URPC 时，UBPU 可以通过 URPC 调用 Entity a 声明的功能或服务。当 UBPU 将访问凭据授权给 Entity b 后，Entity b 可以以同样的方式调用 Entity a 声明的功能或服务。

## 10.3.4 中断

### 10.3.4.1 中断机制

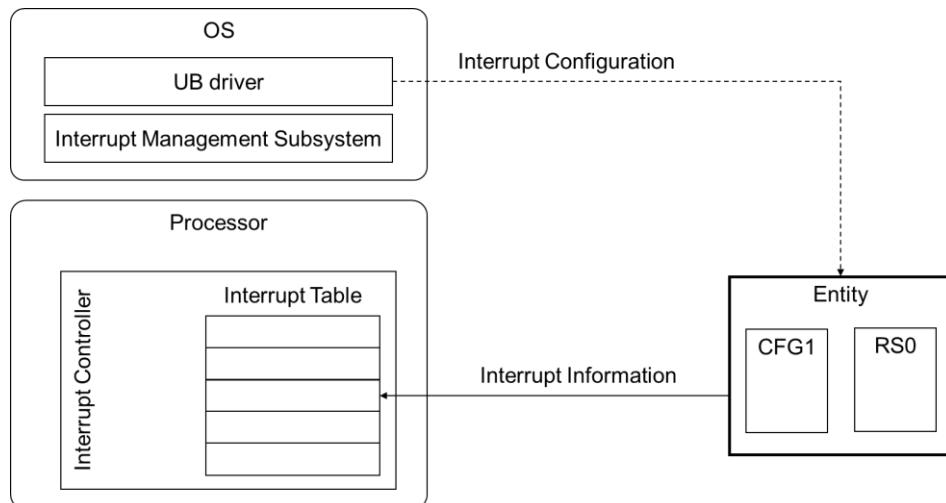


图 10-10 中断机制示意图

UB Controller 或 UB Switch 的配置空间寄存器应描述其是否支持中断功能，其所有 Entity 应同时支持或同时不支持中断功能，其中断功能应对接到系统平台的中断子系统。

UB 驱动为 Entity 向中断管理子系统申请中断，并将中断配置下发给 Entity。Entity 在上报中断时将中断信息写入到中断控制器，中断控制器回调中断管理子系统，中断管理子系统再调用 UB 驱动处理对应的中断。

Entity 产生的 UB Signaled Interrupt ( USI ) 使用 Write 类事务语义承载，USI 消息中的目标内存地址应与中断子系统指定的中断地址一致。

UB 定义了 Type 1 和 Type 2 两种中断实现方式，在实际部署中应选取其中一种使用。

#### 10.3.4.2 Type 1 中断寄存器

表 10-1 Type 1 中断配置寄存器组

Interrupt Enable
Interrupt Number
Interrupt Enable Number
Interrupt Data
Interrupt Address
Interrupt ID
Interrupt Mask
Interrupt Pending

CFG1\_CAP Bitmap 寄存器指示 Entity 是否支持 Type 1 中断，若支持则 CFG1 应实现上表所示的 Type 1 中断配置寄存器组，详见 D.4.3 节，各寄存器说明如下：

1. Interrupt Enable：软件配置该寄存器控制 Entity 中断功能是否启用。
2. Interrupt Number：指示 Entity 支持的中断向量数量。
3. Interrupt Enable Number：软件将 Entity 启用的中断向量数量配置到该寄存器。
4. Interrupt Data：软件将中断向量的起始编号配置到该寄存器，Entity 上报中断时 USI 消息应携带对应的中断向量编号，规则如下：
  - (1) 当启用 1 个中断向量时，Entity 上报的中断数据为 X，X 为配置到 Interrupt Data 寄存器的值。
  - (2) 当启用 N 个中断向量时，Entity 上报的中断数据为[X, X+N-1]，取值与中断向量顺序对应，X 为配置到 Interrupt Data 寄存器的值
5. Interrupt Address：软件将中断子系统分配的中断地址配置到该寄存器，Entity 上报的中断信息应写入到该地址指示的内存中。Entity 发送 USI 消息时，使用 Interrupt Address 寄存器的值作为 UBA，DEID 使用 CFG0 的 UEID 寄存器值，TokenID 使用 CFG1 的 DEV\_TOKEN\_ID 寄存器值。
6. Interrupt ID：软件将中断子系统分配的中断标识配置到该寄存器，Entity 上报中断时 USI 消息应携带 Interrupt ID。

7. Interrupt Mask: 软件配置该寄存器控制 Entity 对应的中断上报是否被屏蔽, 32 bits 分别对应 32 个中断向量编号, bit 0 描述中断向量 0, bit 1 描述中断向量 1, 依次类推。当中断向量没有启用时, 该寄存器对应的 bit 无实际意义。
8. Interrupt Pending: 指示对应的中断向量上报状态, 32 bits 分别对应 32 个中断向量编号, bit 0 描述中断向量 0, bit 1 描述中断向量 1, 依次类推。当中断向量没有启用时, 该寄存器对应的 bit 无实际意义。

### 10.3.4.3 Type 2 中断寄存器

#### 寄存器结构

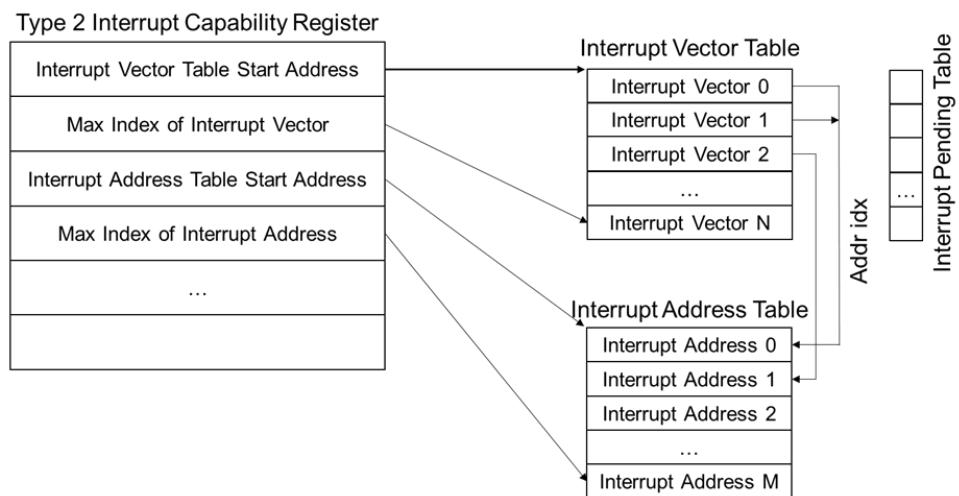


图 10-11 Type 2 寄存器结构示意图

如上图所示, Type 2 中断寄存器分为 4 组, 中断能力寄存器组确定中断向量表和中断地址表的起始地址及表项数量, 每个中断向量都通过 Address Index 指向对应的中断地址, 中断 Pending 表与中断向量表对应, 每个 bit 分别指示对应中断向量的上报状态。

#### 中断能力寄存器组

表 10-2 Type 2 中断能力寄存器组

Max Index of Interrupt Address
Max Index of Interrupt Vector
Interrupt Vector Table Start Address
Interrupt Address Table Start Address
Interrupt Pending Table Start Address
Interrupt ID
Interrupt Mask
Interrupt Enable

CFG1\_CAP Bitmap 寄存器指示 Entity 是否支持 Type 2 中断，若支持则 CFG1 应实现上表所示的 Type 2 中断能力寄存器组，详见 D.4.4 节，各寄存器说明如下：

1. Max Index of Interrupt Address：指示中断地址的最大索引，Entity 支持配置的中断地址数量为 M+1, M 为该寄存器的值；
2. Max Index of Interrupt Vector：指示中断向量的最大索引，Entity 支持的中断向量数量为 N+1, N 为该寄存器的值；多个中断向量可使用相同的中断地址。
3. Interrupt Vector Table Start Address：指示中断向量表的起始地址；
4. Interrupt Address Table Start Address：指示中断地址表的起始地址；
5. Interrupt Pending Table Start Address：指示中断 Pending 表的起始地址；
6. Interrupt ID：软件将中断子系统分配的中断标识配置到该寄存器，Entity 上报中断时 USI 消息应携带 Interrupt ID。
7. Interrupt Mask：软件配置该寄存器控制 Entity 的所有中断上报是否被屏蔽。
8. Interrupt Enable：软件配置该寄存器控制 Entity 中断功能是否启用。

### 中断地址表

表 10-3 Type 2 中断地址表结构

Address Index	Entry Structure					
	64 bits	11 bits	1 bit	20 bits	128 bits	32 bits
0	Interrupt Address	RSVD	Valid	TokenID	DEID	RSVD
1	Interrupt Address	RSVD	Valid	TokenID	DEID	RSVD
...	...	...	...	...	...	...
M	Interrupt Address	RSVD	Valid	TokenID	DEID	RSVD

中断地址表存放在 Entity 资源空间 0 中，每个表项占用 32 Bytes，由 Address Index 索引，结构如上表所示，各字段定义如下：

1. Valid：表项有效位，1 为有效，0 为无效；
2. DEID：中断处理节点的身份标识；
3. TokenID：中断处理节点赋予的访问凭证，Entity 发送 USI 消息时，应在消息中携带该凭证；
4. Interrupt Address：软件将中断子系统分配的中断地址配置到该字段，Entity 发送 USI 消息时，使用 Interrupt Address 字段的值作为 UBA。

软件应按照 4 Bytes 对齐方式更新中断地址表配置，将表项中 Valid 域置为无效后再更新其它字段。

## 中断向量表

表 10-4 Type 2 中断向量表结构

Vector Index	Entry Structure			
	32 bits	15 bits	1 bit	16 bits
0	Interrupt Data	RSVD	Mask	Address Index
1	Interrupt Data	RSVD	Mask	Address Index
...	...	...	...	...
N	Interrupt Data	RSVD	Mask	Address Index

中断向量表存放在 Entity 资源空间 0 中，每个表项占用 8 Bytes，由 Vector Index 索引，结构如上表所示，各域段定义如下：

1. Mask：中断屏蔽状态，1 为屏蔽，0 为不屏蔽；
2. Interrupt Data：软件将中断向量的编号配置到该域段，Entity 上报中断时 USI 消息应携带对应的中断向量编号；
3. Address Index：中断地址索引，不同的中断向量可使用相同的中断地址索引。

软件应按照 4 Bytes 对齐方式更新中断向量表配置，确认 Mask 位有效后再更新其它域段，避免更新正在使用中的表项。

## 中断 Pending 表

中断 Pending 表存在 Entity 资源空间 0 中，表结构是一个 bitmap。每个 bit 对应中断向量表的一个表项，且为顺序对应关系，标识对应中断的 Pending 状态，为 1 时表示当前中断为 Pending 状态。

中断 Pending 状态清除可通过以下方式：

1. 软件配置清除中断屏蔽状态时，Entity 上报中断后，清除 Pending 状态。
2. Entity 复位后 Pending 状态自动清除。

### 10.3.4.4 中断消息

Byte0								Byte1								Byte2								Byte3							
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
Interrupt ID																															
Interrupt Data																															

图 10-12 USI 消息 PDU 格式

Interrupt ID：标识上报中断的 Entity 身份，Entity 发送 USI 消息时使用 CFG1 中 Interrupt ID 寄存器的值。

Interrupt Data: 标识上报的中断向量编号, Entity 发送 USI 消息时生成规则详见 10.3.4.2 节和 10.3.4.3 节中断寄存器描述。

## 10.3.5 消息通知

### 10.3.5.1 本地事件通知

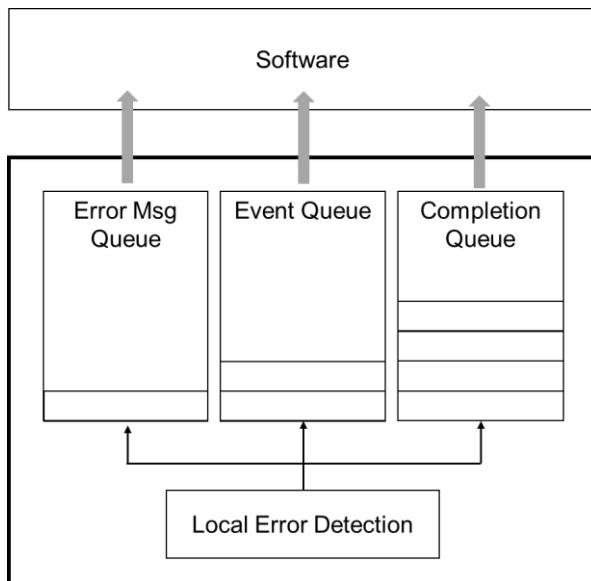


图 10-13 本地事件通知机制

如上图所示，当检测到本地发生错误时，Entity 支持根据错误类型（见 10.6.2 节）将错误事件通过不同的队列上送给软件处理，包括以下方式：

1. Completion Queue

当本地产生 A 类错误需要处理时，Entity 通过 Completion Queue 上报，在 CQE 中填写错误的编码，用于通知软件某个事务出现异常。

2. Event Queue

当本地产生 B 类错误需要处理时，Entity 通过 Event Queue 上报。此方式常用于找不到对应 CQ 的异常事件的上报。

3. Error Message Queue

当本地产生 C 类错误时，Entity 0 应通过 Error Message Queue 上报，上报流程如下图所示。

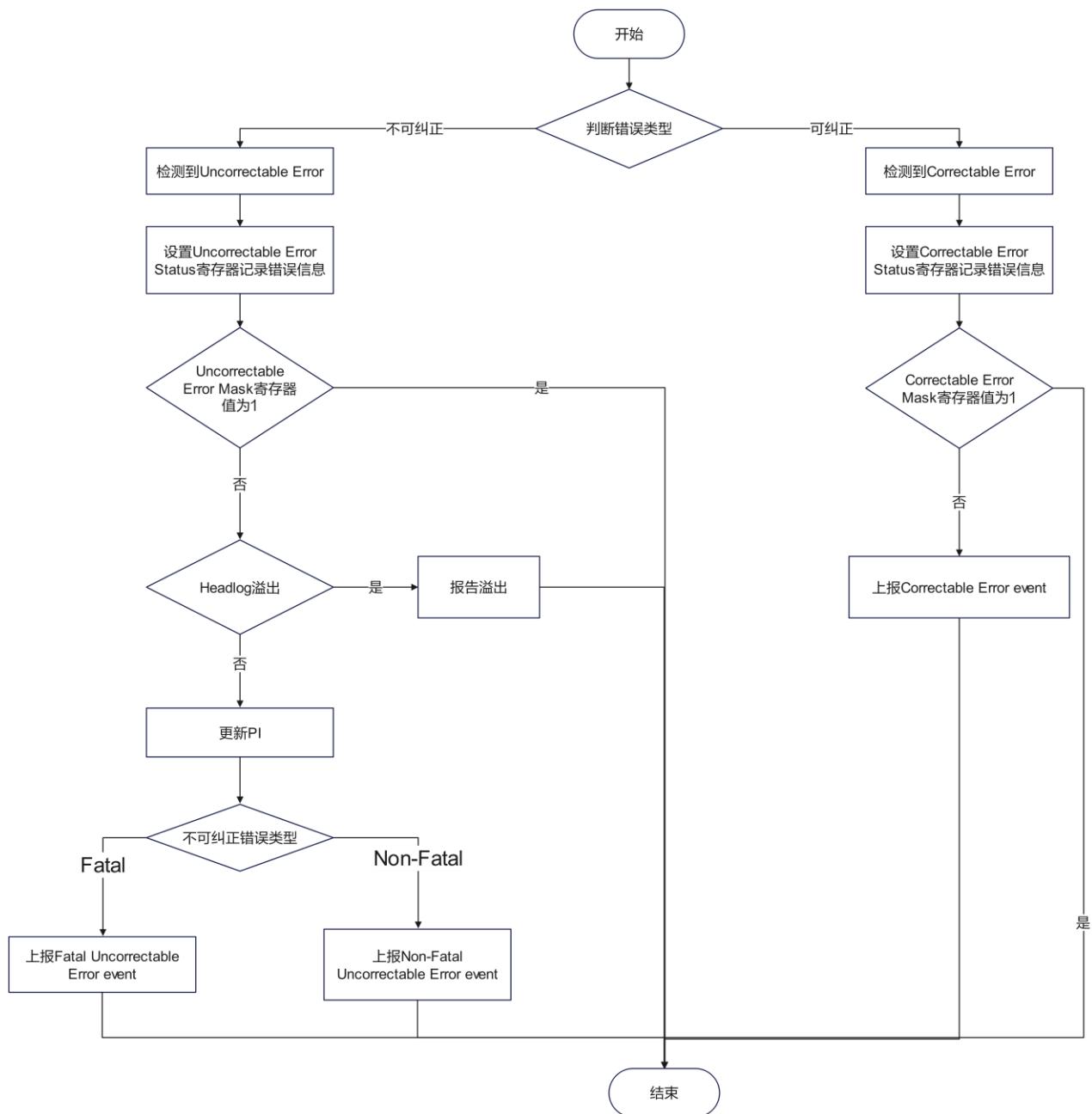


图 10-14 Error Message 事件上报流程示意图

### 10.3.5.2 远端事件通知

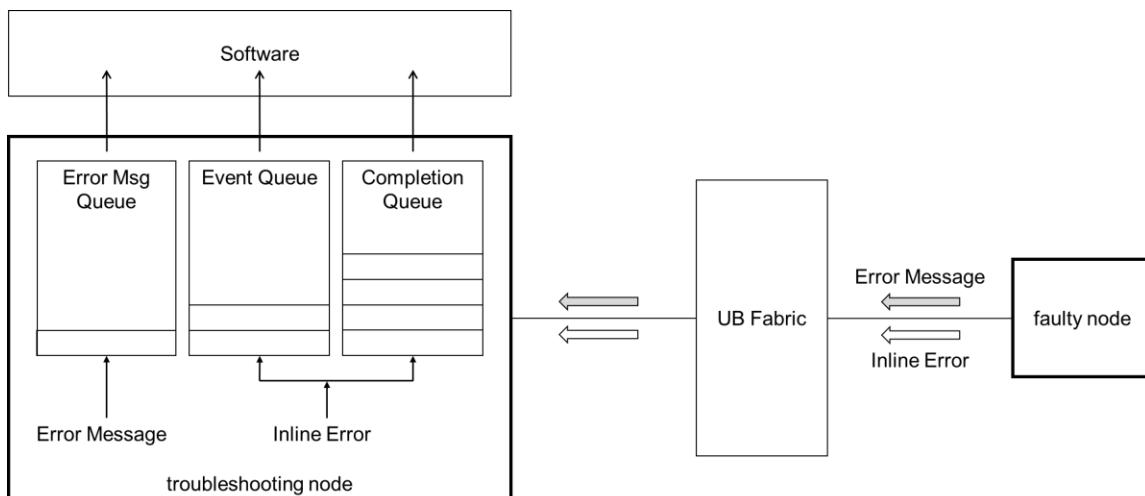


图 10-15 远端事件通知流程示意图

如上图所示，当检测到发生错误时，UB 支持根据错误类型将错误事件通过不同方式传递到故障处理节点，包括以下方式：

#### 1. Inline Error

##### (1) 响应包携带错误信息：

Target 在处理事务请求过程中发生错误时，可将错误信息携带在响应包的 Status 域段内返回到 Initiator，通知 Initiator 其事务请求发生了错误。

##### (2) 数据包携带 Poison 指示：

使用 BTAH 的 Poison 域段指示数据包携带的数据是否存在错误，标记 Poison 的数据在到达目的端后不能被视为正常数据进行处理。

#### 2. Error Message

当产生 C 类错误时，Entity 0 可发送 Error Message 将错误事件传递到 UBFM 处理。

## 10.4 管理机制

### 10.4.1 配置空间

#### 10.4.1.1 访问权限控制

配置空间是 Entity 开放给 UBFM 和具有其使用权的 UBPU 的配置管理接口，存储设备的能力、状态、配置等信息。具有 Entity 使用权的 UBPU 拥有其部分管理权限，简称为 User。UBFM 与 User 的管理权限不同，说明如下：

1. CFG0: UBFM 具有读写权限, User 仅具有读权限, 不具有写权限。
  - (1) 管理消息的 NTH.Mgmt 为 1, 表示该管理命令由 UBFM 发出, 具有读写权限。
  - (2) 管理消息的 NTH.Mgmt 为 0, 表示该管理命令由 User 发出。管理消息的 UPI 匹配 Entity 的 UPI 时, 具有读权限, 不具有写权限。管理消息的 UPI 与 Entity 的 UPI 不匹配时, 则为非法命令。
2. CFG1: UBFM 具有读写权限, User 具有读写权限。
  - (1) 管理消息的 NTH.Mgmt 为 1, 表示该管理命令由 UBFM 发出, 具有读写权限。
  - (2) 管理消息的 NTH.Mgmt 为 0, 表示该管理命令由 User 发出。管理消息的 UPI 匹配 Entity 的 UPI 时, 具有读写权限。管理消息的 UPI 与 Entity 的 UPI 不匹配时, 则为非法命令。

#### 10.4.1.2 访问机制

系统软件可通过本地 UB Controller (如总线控制器, 见 C.3.2 节) 发送配置空间访问请求消息, 远端 UB Controller 和 UB Switch 解析并执行配置管理命令, 返回响应消息。本地 UB Controller 将接收到的响应消息上送给系统软件处理。

如下图所示, 系统软件访问配置空间时可并发多个访问请求, 每个请求消息和响应消息通过 MSN 一一对应, 即系统软件收到响应消息后, 通过 MSN 来匹配对应的访问请求。

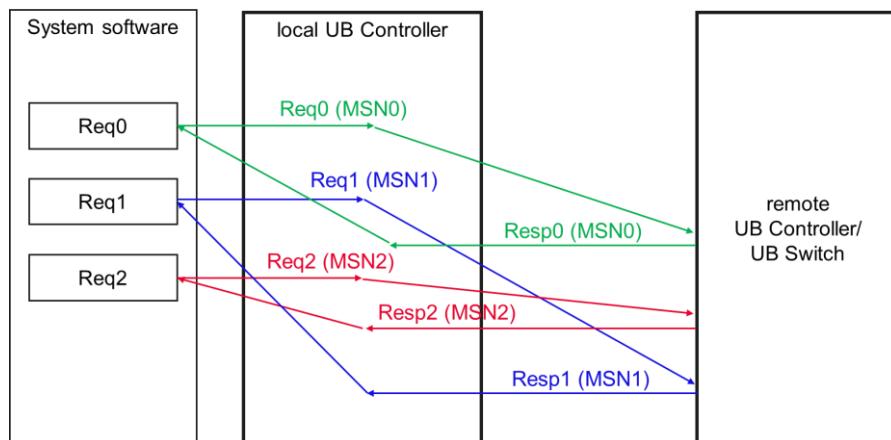


图 10-16 配置空间访问机制示意图

#### 10.4.1.3 配置空间结构

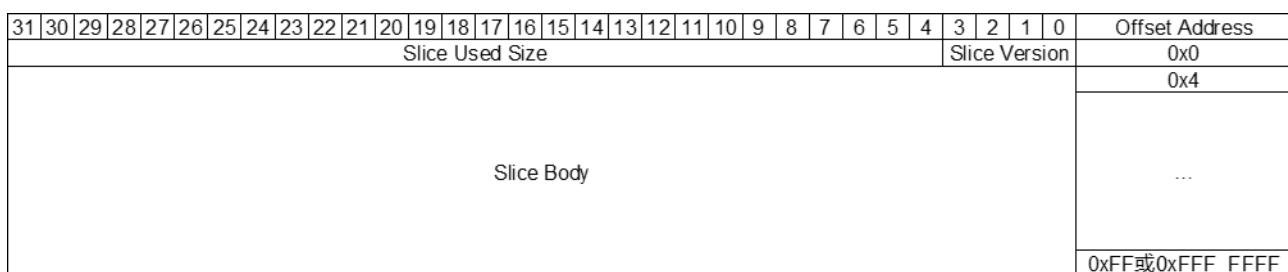


图 10-17 配置空间 Slice 结构示意图

本规范将配置空间分为若干片区间,每片区间称为 Slice,存在强相关性的寄存器被定义在同一个 Slice 内。如上图所示, Slice 由 Header 和 Body 组成, Header 定义版本号和该 Slice 实际占用的地址空间, Body 内排布该 Slice 的寄存器, Offset Address 的单位为 4 Bytes。Slice 的地址空间为 1 KB 或 1 GB, ROUTE TABLE Slice 的地址空间为 1 GB, 其余 Slice 的地址空间固定为 1 KB。

表 10-5 Slice Header 寄存器定义

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Slice Used Size	31:4	指示该 Slice 实际占用的地址空间, 单位为 4 Bytes	RO	0x0	-
Slice Version	3:0	Slice 的版本号	RO	0x0	-

访问 Slice 中未定义的地址空间时, 即访问偏移地址位于[X,0xFF]或[X,0xFFFF]范围内的地址空间, 返回全 0 值, 其中 X 为 Slice Used Size 寄存器的值。

表 10-6 配置空间 Slice 的地址空间定义

Category	Start Address	End Address	CAP Name
CFG0_BASIC	0x0000_0000	0x0000_00FF	CFG0_BASIC
CFG0_CAP	0x0000_0100	0x0000_01FF	CFG0_CAP1_RSVD
	0x0000_0200	0x0000_02FF	CFG0_CAP2_SHP
	0x0000_0300	0x0000_03FF	CFG0_CAP3_DEVICE_ERR_RECORD
	0x0000_0400	0x0000_04FF	CFG0_CAP4_DEVICE_ERR_INFO
	0x0000_0500	0x0000_05FF	CFG0_CAP5_EMQ
	...	...	
	0x0000_FF00	0x0000_FFFF	
CFG1_BASIC	0x0001_0000	0x0001_00FF	CFG1_BASIC
CFG1_CAP	0x0001_0100	0x0001_01FF	CFG1_CAP1_DECODER
	0x0001_0200	0x0001_02FF	CFG1_CAP2_JETTY
	0x0001_0300	0x0001_03FF	CFG1_CAP3_INT_TYPE1
	0x0001_0400	0x0001_04FF	CFG1_CAP4_INT_TYPE2
	0x0001_0500	0x0001_05FF	CFG1_CAP5_RSVD
	0x0001_0600	0x0001_06FF	CFG1_CAP6_UB_MEM
	...	...	
	0x0001_FF00	0x0001_FFFF	

<b>Category</b>	<b>Start Address</b>	<b>End Address</b>	<b>CAP Name</b>
CFG0_PORT0_BASIC	0x0002_0000	0x0002_00FF	CFG0_PORT0_BASIC
CFG0_PORT0_CAP	0x0002_0100	0x0002_01FF	CFG0_PORT_CAP1_LINK
	0x0002_0200	0x0002_02FF	CFG0_PORT_CAP2_LINK_LOG
	0x0002_0300	0x0002_03FF	CFG0_PORT_CAP3_RSVD
	0x0002_0400	0x0002_04FF	CFG0_PORT_CAP4_DATA_RATE1
	0x0002_0500	0x0002_05FF	CFG0_PORT_CAP5_DATA_RATE2
	0x0002_0600	0x0002_06FF	CFG0_PORT_CAP6_DATA_RATE3
	0x0002_0700	0x0002_07FF	CFG0_PORT_CAP7_DATA_RATE4
	0x0002_0800	0x0002_08FF	CFG0_PORT_CAP8_DATA_RATE5
	0x0002_0900	0x0002_09FF	CFG0_PORT_CAP9_DATA_RATE6
	0x0002_0A00	0x0002_0AFF	CFG0_PORT_CAP10_DATA_RATE7
	0x0002_0B00	0x0002_0BFF	CFG0_PORT_CAP11_DATA_RATE8
	0x0002_0C00	0x0002_0CFF	CFG0_PORT_CAP12_DATA_RATE9
	0x0002_0D00	0x0002_0DFF	CFG0_PORT_CAP13_RSVD
	0x0002_0E00	0x0002_0EFF	CFG0_PORT_CAP14_EYE_MONTIOR
	0x0002_0F00	0x0002_0FFF	CFG0_PORT_CAP15_QDLWS
CFG0_PORT1_BASIC	0x0002_1000	0x0002_10FF	CFG0_PORT_CAP16_SRIS
	0x0002_1100	0x0002_11FF	CFG0_PORT_CAP17_RSVD
	0x0002_1200	0x0002_12FF	CFG0_PORT_CAP18_LINK_PERF
	0x0002_1300	0x0002_13FF	CFG0_PORT_CAP19_LINK_ERR_INJECTION
	0x0002_1400	0x0002_14FF	CFG0_PORT_CAP20_LTSM_ST
	0x0002_1500	0x0002_15FF	CFG0_PORT_CAP21_PORT_ERR_RECORD
	...	...	
	0x0002_FF00	0x0002_FFFF	
CFG0_PORT1_BASIC	0x0003_0000	0x0003_00FF	CFG0_PORT1_BASIC
CFG0_PORT1_CAP	0x0003_0100	0x0003_01FF	CFG0_PORT1_CAP1_LINK
	0x0003_0200	0x0003_02FF	CFG0_PORT1_CAP2_LINK_LOG
	...	...	
	0x0003_FF00	0x0003_FFFF	
...	...	...	
CFG0_ROUTE_TABLE	0xF000_0000	0xFFFF_FFFF	CFG0_ROUTE_TABLE

Start Address 和 End Address 的单位为 4 字节。

CFG0\_BASIC 中 CFG0\_CAP Bitmap 寄存器指示哪些 CFG0 CAP 存在。

CFG1\_BASIC 中 CFG1\_CAP Bitmap 寄存器指示哪些 CFG1 CAP 存在。

CFG0\_PORT\_BASIC 中 PORT\_CAP Bitmap 寄存器指示哪些 PORT CAP 存在。

CFG0\_BASIC 中 Route Table Supported 寄存器指示 CFG0\_ROUTE\_TABLE 是否存在。

以上 Slice 中的具体寄存器列表见附录 D。

#### 10.4.1.4 配置空间寄存器属性

表 10-7 配置空间寄存器属性定义

属性	属性描述
RO	Read-Only 只读
RW	Read-Write 可读可写
RW1C	Write-1-to-clear-status 可读可写，通过写 1 将寄存器清 0
HwInit	Hardware Initialized 上电初始化完成后，寄存器值固定不变，寄存器属性变为 RO。 仅设备级复位可以复位该类型寄存器。
ROS	Sticky-Read-Only，仅设备级复位可以复位该类型寄存器。
RWS	Sticky-Read-Write，仅设备级复位可以复位该类型寄存器。
RW1CS	Sticky-Write-1-to-clear-status，仅设备级复位可以复位该类型寄存器。
*_DE0_EO	*表示前述寄存器属性，*_DE0_EO 表示： 寄存器位于 Entity 0 时，寄存器属性为*； 寄存器位于 Entity n (n 为非 0)时，寄存器不存在。
*_DEN_RO	*表示前述寄存器属性，*_DEN_RO 表示： 寄存器位于 Entity 0 时，寄存器属性为*； 寄存器位于 Entity n (n 为非 0)时，寄存器属性为 RO。

访问配置空间中未定义的寄存器或寄存器字段时，统一按如下规则处理：

1. 写操作不生效，返回命令执行成功响应；
2. 读操作返回命令执行成功响应，返回数据为全 0；

### 10.4.2 资源空间

资源空间是 Entity 开放给软件操作的配置管理接口，用于存放中断信息、功能或服务相关的配置信息和厂商自定义信息的设备存储区域。每个 Entity 的资源空间最多可分成三段，通过 CFG1\_BASIC 中的 ERS 寄存器组进行配置。资源空间整体布局如下图所示：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
																															ERS0_SA	
																															ERS0_SA+0x4	
																															...	
																															Reserved	
																															ERS1_SA	
																															ERS1_SA+0x4	
																															...	
																															Reserved	
																															ERS2_SA	
																															ERS2_SA+0x4	
																															...	

图 10-18 资源空间整体布局

当支持 Type 2 中断时，资源空间 0 如下图所示，用于存放中断向量表、中断地址表和中断 Pending 表信息。资源空间 1 和资源空间 2 是 Entity 开放给设备驱动软件的配置接口，由厂商自定义。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Start Address (4 Byte)
																															Interrupt Vector Table Start Address	
																															Interrupt Vector Table Start Address + 0x1	
																															Interrupt Vector Table Start Address + 0x2	
																															Interrupt Vector Table Start Address + 0x3	
																															...	
																															Interrupt Vector Table Start Address + (N-1)*0x2	
																															Interrupt Vector Table Start Address + (N-1)*0x2 + 0x1	
																															Interrupt Address Table Start Address	
																															Interrupt Address Table Start Address + 0x1	
																															...	
																															Interrupt Address Table Start Address + 0x7	
																															Interrupt Address Table Start Address + 1*0x8	
																															Interrupt Address Table Start Address + 1*0x8 + 0x1	
																															...	
																															Interrupt Address Table Start Address + 1*0x8 + 0x7	
																															...	
																															Interrupt Address Table Start Address + (M-1)*0x8	
																															Interrupt Address Table Start Address + (M-1)*0x8 + 0x1	
																															...	
																															Interrupt Address Table Start Address + (M-1)*0x8 + 0x7	
																															Interrupt Pending Table Start Address	
																															Interrupt Pending Table Start Address + 0x1	
																															...	
																															Interrupt Pending Table Start Address + ((N+31)/32)*0x1	

图 10-19 资源空间 0

User 可将 Entity 资源空间以 MMIO 方式映射到自身的内存地址空间中，User 为所有 Entity 分配的地址空间范围不能重叠。User 可以通过基于 LPH.CFG=6 数据包承载的 Read/Write 操作实现对 Entity 资源空间的访问，数据包携带 User 侧 UPI，Entity 在接收时校验 UPI，UPI 不匹配则为非法命令。

### 10.4.3 管理命令

#### 10.4.3.1 管理消息

管理命令用于 UB Controller 和 UB Switch 的发现和管理，包括链路邻居通告、枚举管理和配置管理，其中配置管理包括管理命令下发和事件上报。管理命令在 UB 链路上使用管理消息承载。

枚举管理消息格式如下：

LPH	NTH	UPIH	Padding	Scan Header	Scan PDU	ICRC
-----	-----	------	---------	-------------	----------	------

图 10-20 枚举管理消息格式

1. LPH.CFG = 6;
2. NTH.Mgmt: UBFM 发起的命令该域段值为 1;
3. NTH.SCNA: 未分配 CNA 时，该域段为 0;
4. NTH.DCNA: 未分配 CNA 时，该域段为 0;
5. NTH.NLP = 3'b001;
6. UPIH: 16-bit UPI 包头，低 15 bits 为 UPI 值，UBFM 发起的命令该域段值为 0x7FFF，0x7FFF 为协议保留给 UBFM 管理使用;
7. Padding: 该域段长度 2 Bytes，填充为 0;
8. Scan Header: 详见 10.4.3.2 节枚举管理;
9. Scan PDU: 详见 10.4.3.2 节枚举管理;
10. ICRC: 系统管理员可在上电启动配置中指定是否启用 ICRC 保护;
11. 枚举管理消息的 Scan Header 和 Scan PDU 的总长度≤1024 Bytes，一个枚举管理消息只能承载在一个数据包内。
12. 枚举管理消息没有 TAACK 确认，不能保证传输的可靠性，由管理软件负责可靠性处理。

配置管理消息格式如下：

LPH	NTH	CTPH	UPIH	SEID	DEID	BTAH	MGMTETAH	PDU	ICRC
-----	-----	------	------	------	------	------	----------	-----	------

图 10-21 配置管理消息格式

1. LPH.CFG = 6;
2. NTH.Mgmt: UBFM 发起的命令该域段值为 1;
3. NTH.DCNA: 下发管理命令时为目标管理对象的地址；事件上报时为 UBFM 或 User 的地址。
4. NTH.SCNA: 下发管理命令时为 UBFM 或 User 的地址；事件上报时为产生该事件的节点地址。
5. NTH.NLP = 0;
6. UPIH: 16-bit UPI 包头，低 15 bits 为 UPI 值，UBFM 发起的命令该域段值为 0x7FFF;User 发起的配置管理消息该域段值为 User 对应的 UPI 值。

7. BTAHOpcode = 0x10;
8. 配置管理消息的 PDU 长度≤1024 Bytes，一个配置管理消息只能承载在一个数据包内；
9. 配置管理消息没有 TAACK 确认，不能保证传输的可靠性，由管理软件负责可靠性处理。

枚举和配置管理命令响应的错误码定义如下：

1. 0x00: 正常完成；
2. 0x01: 管理越权；
3. 0x02: 管理目标不存在；
4. 0x03: 消息码错误；
5. 0x04: 消息长度错误；
6. 0x05~0x1F: 保留；
7. 0x20: 资源不足；
8. 0x21: 权限不足；
9. 0x22: 地址错误；
10. 0x23: 忙碌；
11. 0x24: 对象已存在；
12. 0x25: 对象不存在；
13. 0x26: 参数非法；
14. 0x27: 执行错误；
15. 0x28~0xFE: 保留；
16. 0xFF: 未知错误。

### 10.4.3.2 枚举管理

#### Scan Header

UBFM 在发起枚举时，UB Controller 和 UB Switch 还未被配置 CNA，枚举消息不具备通过路由转发直达的能力。枚举管理消息采用源路由进行转发，Scan Header 作为枚举管理消息的源路由扩展网络头，格式如下：

Byte0								Byte1								Byte2								Byte3															
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								
RSVD								R	Type	Hops								Step																					
Forward Path																																							
Return Path																																							

图 10-22 Scan Header 格式

UB Controller 和 UB Switch 收到枚举管理消息后，根据 Scan Header 的指示进行转发或响应，各域段定义如下：

1. R：长度为 1 bit，指示 Scan HEADER 是否携带 Return path，1：携带；0：未携带。  
注：请求消息总是携带 Return path，响应消息总是不携带 Return path。目的节点采用请求消息中的 Return Path 作为回复响应消息的 Forward Path。
2. Type：长度为 4 bits，指示 Forward Path 和 Return Path 的出端口域段位宽。
  - (1) 0：位宽为 4 bits；
  - (2) 1：位宽为 8 bits；
  - (3) 2：位宽为 16 bits；
  - (4) 3~15：预留。
3. Hops：长度为 8 bits，指示 Forward Path 和 Return Path 包含的跳数，Forward 和 Return Path 的跳数相同。当 Hops 为 0 时表示 Scan Header 未携带 Forward Path 信息，可用于 UBFM 查找本节点的设备信息；当 Hops 大于 0 时表示 Scan Header 携带 Forward Path 信息。
4. Step：长度为 8 bits，指示当前节点转发枚举管理消息的出端口在 Forward Path 出端口列表中的位置。Step 初始值为 0，从接收端口上收到枚举管理消息后首先将 step 加 1；当 Step 与 Hops 相等时则表示已到达目的节点，当 Step 小于 Hops 时则表示需继续转发；
5. Forward Path：指示 Forward 方向转发路径的出端口列表，出端口域段的位宽由 Type 决定，Forward Path 有效长度为 Hops\*出端口域段位宽，Forward Path 总长度 4 Bytes 对齐，不足 4 Bytes 则用 Padding 补齐。将 Forward Path 看作一个数组，当前节点的出端口由 Forward Path[Step]决定。
6. Return Path：指示目的端回复响应消息时的反向出端口列表，每个出端口的位宽由 Type 决定，Return Path 有效长度为 Hops\*出端口域段位宽，Return Path 总长度 4 Bytes 对齐，不足 4 Bytes 则用 Padding 补齐。

### Scan PDU

Scan PDU 承载枚举管理命令，首 DW 定义如下：

Byte 0								Byte 1								Byte 2								Byte 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
Version								CMD								...								...							

图 10-23 Scan PDU 首 DW 格式

Scan PDU 的前 2 个字节为命令头，用以识别不同的管理命令。

1. Version：长度为 8 bits，版本号，当前版本号为 1
2. CMD：长度为 8 bits，管理命令字，支持用户在自定义编码区定制命令

表 10-8 管理命令字

CMD 编码	管理命令字描述
0	拓扑查询
1	配置 CNA
2	查询 CNA
3~255	预留编码

### 拓扑查询

#### 1. 拓扑查询请求

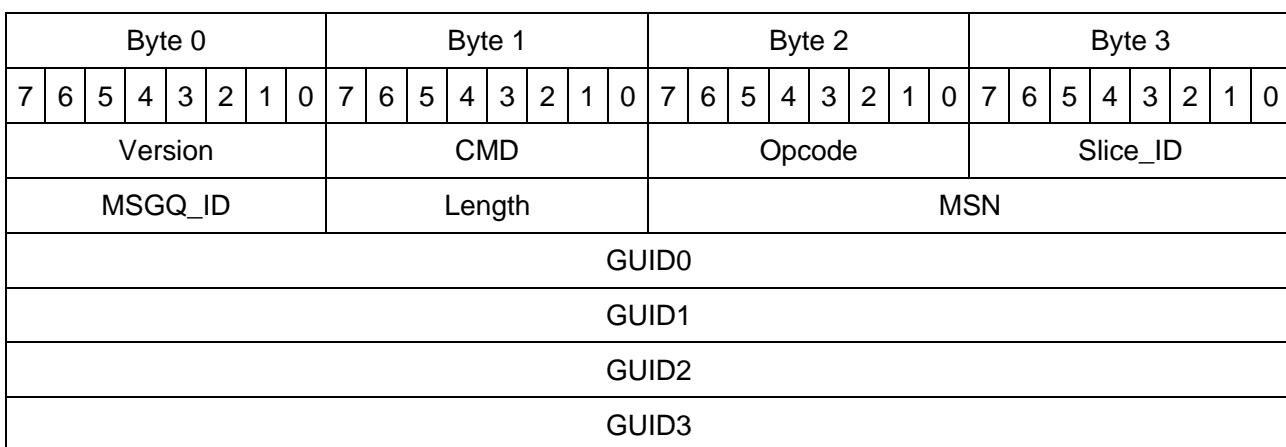


图 10-24 拓扑查询请求 Scan PDU 格式

UBFM 发送拓扑查询请求，拓扑查询请求 Scan PDU 格式由上表所示：

- (1) Version: 值为 1;
- (2) CMD: 值为 0;
- (3) Opcode: 长度为 8 bits, 操作码, 值为 1: 拓扑查询请求。
- (4) Slice\_ID: 长度为 8 bits, 本次拓扑查询请求所请求的设备信息分片 ID, Slice\_ID 从 0 开始编码。
- (5) MSGQ\_ID: 长度为 8 bits, 发送该命令请求的消息队列号, 由本地 UB Controller 为该域段赋值。
- (6) Length: 域段长度为 8 bits, 指示该消息携带的 PDU 长度, 单位是 4 字节。
- (7) MSN: 长度为 16 bits, 本消息的序号, 该序号是单调递增的序列号, 用于 UBFM 识别其发起的并发消息。
- (8) GUID: 长度为 128 bits, 目的节点的 GUID。

## 2. 拓扑查询响应

Byte 0								Byte 1								Byte 2								Byte 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																								
Version								CMD								Opcode								Status																															
MSGQ_ID								Length								MSN																																							
GUID0																																																							
GUID1																																																							
GUID2																																																							
GUID3																																																							
Resp_content																																																							

图 10-25 拓扑查询响应 Scan PDU 格式

目的节点收到拓扑查询请求后收集本节点的设备信息，向 UBFM 返回拓扑查询响应。拓扑查询响应 Scan PDU 格式由上表所示：

- (1) Version: 值为 1;
- (2) CMD: 0;
- (3) Opcode: 长度为 8 bits, 操作码, 0: 拓扑查询响应。
- (4) Status: 长度为 8 bits, 完成状态, 0: 成功; 1~255: 失败, 值标识错误码, 错误码定义见 10.4.3.1 节。
- (5) MSGQ\_ID: 长度为 8 bits, 返回拓扑查询请求携带的 MSGQ\_ID。
- (6) Length: 长度为 8 bits, 指示该消息携带的 PDU 长度, 单位是 4 字节。
- (7) MSN: 长度为 16 bits, 返回拓扑查询请求携带的 MSN。
- (8) GUID: 长度为 128 bits, 本节点的 GUID。
- (9) Resp\_content: 拓扑查询响应的具体内容由多个连续 TLV 组成, TLV 结构如下:

Byte 0								Byte 1								Byte 2								Byte 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
Type								Length								Value															
Value																															

图 10-26 拓扑查询响应的 TLV 结构

- Type: TLV 信息类型 (详细定义见下述列表), 决定 Value 域段如何解析。
- Length: 该 TLV 的总长度, 单位是字节, Length 要求是 4 的整数倍。
- Value: 该 TLV 携带的信息。

表 10-9 拓扑查询响应的 TLV 类型

Type	类型	定义	说明
0	Slice_info	拓扑查询响应的分片信息	必选
1	Port_num	端口数量信息	必选
2	Port_info	端口信息	必选
3	RSVD	本规范保留	/
4	Cap_info	设备能力和配置信息	必选
5	RSVD	本规范保留	/
6	RSVD	本规范保留	/
7~127	RSVD	本规范保留	/
128~255	Vendor defined	厂商自定义	可选

目的节点可以根据需要选择响应消息中携带的 TLV 类型和数量，下述为响应消息各类型 TLV 的详细定义：

Byte 0								Byte 1								Byte 2								Byte 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
0								4								Total_slice								Slice_ID							

图 10-27 分片信息

若设备信息总量大于一个拓扑查询响应消息的承载能力时，目的节点应将信息划分为固定的分片，并根据 UBFM 的查询请求返回指定分片的内容。

- Total\_slice：长度为 8 bits，设备信息分片的总数。
- Slice\_ID：长度为 8 bits，本次拓扑查询响应返回的设备信息分片 ID。

Byte 0								Byte 1								Byte 2								Byte 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
1								8								Total_num_ports															
Num_port_TLV																RSVD															

图 10-28 端口数量信息

- Total\_num\_ports：长度为 16 bits，目的节点的端口总数。
- Num\_port\_TLV：长度为 16 bits，本拓扑查询响应携带的端口信息 TLV 的数量。

Byte 0								Byte 1								Byte 2								Byte 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																																
2								28								RSVD		T	W	B	S	RSVD																																									
Local_port_idx																Remote_port_idx																																															
RSVD																																																															
Remote_GUID0																																																															
Remote_GUID1																																																															
Remote_GUID2																																																															
Remote_GUID3																																																															

图 10-29 端口信息

- T: 长度为 1 bit, 端口类型, 0: Port; 1: vPort;
- W: 长度为 1 bit, 端口的 work mode 标记, 0: Exclusive work mode, 表示总线控制器独占使用该端口; 1: shared work mode, 表示总线控制器和其他集成控制器共享使用该端口。
- B: 长度为 1 bit, 本地端口是否为 boundary 端口, 0: Non-boundary; 1: Boundary。
- S: 长度为 1 bit, 端口 link 状态, 1 代表 up, 0 代表 down;
- Local\_port\_idx: 长度为 16 bits, 本地端口号;
- Remote\_port\_idx: 长度为 16 bits, 对端端口号。当 S 为 1 时有效。
- Remote\_GUID: 长度为 128 bits, 邻居节点 Entity 0 的 GUID。当 S 为 1 时有效。

Byte 0								Byte 1								Byte 2								Byte 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	
4								8								RSVD		Sup_MTU		RSVD		MTU		RSVD								DA
RSVD																Class Code																

图 10-30 设备能力和配置信息

- Sup\_MTU: 长度为 3 bits, 支持的最大传输层 MTU。
- MTU: 长度为 3 bits, 配置的传输层 MTU 大小。
- DA: device authentication, 长度为 1 bit, 是否支持认证能力, 0: 不支持; 1: 支持。
- Class Code: Entity 0 的配置空间 1 中 Class Code 信息

## 配置 CNA

### 1. 配置 CNA 请求

Byte 0								Byte 1								Byte 2								Byte 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																
Version								CMD								Opcode								RSVD																							
MSGQ_ID								Length								MSN																															
GUID0																GUID1																															
GUID2																GUID3																															
Port_idx																RSVD																															
RSVD								CNA																																							

图 10-31 配置 CNA 请求 Scan PDU 格式

配置 CNA 命令用于 UBFM 下发 CNA 地址配置，配置 CNA 请求 Scan PDU 格式由上表所示，长度应 4 字节对齐：

- (1) CMD: 1。
- (2) Opcode: 操作码，长度为 8 bits
  - 2: 配置 Primary CNA;
  - 3: 配置 Port CNA;
  - 4: 配置 Primary CNA，且同时配置所有端口的 Port CNA 同 Primary CNA;
  - Others: 保留。
- (3) Length: 指示该消息携带的 PDU 长度，单位是 4 字节。
- (4) MSGQ\_ID: 发送该命令请求的消息队列号。
- (5) MSN: 本消息的序号，该序号是单调递增的序列号，用于 UBFM 识别其发起的并发消息。
- (6) GUID: 目的节点的 GUID。
- (7) Port\_idx: 端口索引，Opcode=3 时有效；

### 2. 配置 CNA 响应

Byte 0								Byte 1								Byte 2								Byte 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								
Version								CMD								Opcode								Status															
MSGQ_ID								Length								MSN																							
GUID0																																							

Byte 0								Byte 1								Byte 2								Byte 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
GUID1																															
GUID2																															
GUID3																															

图 10-32 配置 CNA 响应 Scan PDU 格式

目的节点收到配置 CNA 请求后完成地址配置，向 UBFM 返回配置响应。配置 CNA 响应 Scan PDU 格式由上表所示：

- (1) CMD: 1。
- (2) Opcode: 操作码，0: 配置完成确认。
- (3) Status: 完成状态，0: 成功；1~255: 失败，值标识错误码，错误码定义见 10.4.3.1 节。
- (4) Length: 指示该消息携带的 PDU 长度，单位是 4 字节。
- (5) MSGQ\_ID: 返回配置 CNA 请求携带的 MSGQ\_ID。
- (6) MSN: 返回配置 CNA 请求携带的 MSN。
- (7) GUID: 本节点的 GUID。

## 查询 CNA

### 1. 查询 CNA 请求

Byte 0								Byte 1								Byte 2								Byte 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																																
Version								CMD								Opcode								RSVD																																							
MSGQ_ID								Length								MSN																																															
GUID0																																																															
GUID1																																																															
GUID2																																																															
GUID3																																																															
RSVD																Port_idx																																															

图 10-33 查询 CNA 请求 Scan PDU 格式

查询 CNA 命令用于 UBFM 查询网络地址配置，查询 CNA 请求 Scan PDU 格式由上表所示：

- (1) CMD: 2。
- (2) Opcode: 操作码
  - 2: 查询 Primary CNA;
  - 3: 查询 Port CNA;

- Others: 保留。
- (3) Length: 指示该消息携带的 PDU 长度, 单位是 4 字节。
- (4) MSGQ\_ID: 发送该命令请求的消息队列号。
- (5) MSN: 本消息的序号, 该序号是单调递增的序列号, 用于 UBFM 识别其发起的并发消息。
- (6) GUID: 目的节点的 GUID。
- (7) Port\_idx: 端口索引, opcode=3 时有效

## 2. 查询 CNA 响应

Byte 0								Byte 1								Byte 2								Byte 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																								
Version								CMD								Opcode								Status																															
MSGQ_ID								Length								MSN																																							
GUID0																																																							
GUID1																																																							
GUID2																																																							
GUID3																																																							
RSVD								CNA																																															

图 10-34 查询 CNA 响应 Scan PDU 格式

查询 CNA 响应 Scan PDU 格式由上表所示:

- (1) CMD: 2。
- (2) Opcode: 操作码, 0: 查询响应。
- (3) Length: 指示该消息携带的 PDU 长度, 单位是 4 字节。
- (4) MSGQ\_ID: 返回查询 CNA 请求携带的 MSGQ\_ID。
- (5) MSN: 返回查询 CNA 请求携带的 MSN。
- (6) GUID: 本节点的 GUID。
- (7) Status: 完成状态, 0: 成功; 1~255: 失败, 值标识错误码, 错误码定义见 10.4.3.1 节。

### 10.4.3.3 配置管理

#### Management Extended TAH ( MGMTETAH )

表 10-10 MGMTETAH 格式

Byte 0								Byte 1								Byte 2								Byte 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
Sub Code[3:0]	Code[2:0]	RSP	Status[7:0]								RSVD				PDU Length[11:0]																

MGMTETAH 格式如上表所示，各域段定义如下：

1. Sub Code：配置管理命令子码；
2. Code：配置管理命令码；
3. Rsp：命令请求消息或命令响应消息；  
 (1) 0: Request;  
 (2) 1: Response;
4. Status: 命令请求的完成状态, Rsp 为 1 时有效, 该域段值为错误码, 错误码定义见 10.4.3.1 节；
5. PDU Length: 命令消息携带 PDU 的长度, 单位 Byte, 范围支持 0-1024。

**表 10-11 MGMTETAH Code and Sub Code 编码**

<b>Code</b>	<b>Sub Code</b>	<b>Description</b>
3'b000 Error Message	4'b0000	Correctable Error event
	4'b0001	Non-Fatal Uncorrectable Error event
	4'b0010	Fatal Uncorrectable Error event
	4'b0011-4'b1111	Reserved
3'b001 Link Change Message	4'b0000	Link Up event
	4'b0001	Link Down event
	4'b0010	Hot-plug Button Pressed event
	4'b0011	Hot-plug Presence Detect Changed event
	4'b0100-4'b1111	Reserved
3'b010 Configuration Message	4'b0000	Configuration Space0 Read
	4'b0001	Configuration Space0 Write
	4'b0010	Configuration Space1 Read
	4'b0011	Configuration Space1 Write
	4'b0100-4'b1111	Reserved
3'b011 VDM Message	4'b0000	MCTP Over UB
	4'b0001	Unified Virtual Bus
	4'b0010-4'b1110	Reserved for UB Spec
	4'b1111	Reserved for Vendor
3'b100 Entity Information Exchange Message	4'b0000	Reserved
	4'b0001	Resource Space Information Exchange
	4'b0010	Obtaining Entity Information
	4'b0011	Reserved
	4'b0100	Reserved
	4'b0101	Reserved
	4'b0110	Reserved
	4'b0111	Link Neighbor Query

Code	Sub Code	Description
	4'b1000-4'b1111	Reserved
3'b101 Device Security Management Message	4'b0000	Reserved
	4'b0001	Device Authentication Interaction
	4'b0010	Token Check Configuration
	4'b0011	Reserved
	4'b0100	Reserved
	4'b0101-4'b1111	Reserved
3'b110 Pooled Resource Management Message	4'b0000	Entity Registration
	4'b0001	Entity Dereistration
	4'b0010	Bus Instance Create
	4'b0011	Bus Instance Destroy
	4'b0100	Configuration Completion Notification
	4'b0101	Port Reset Notification
	4'b0110-4'b1111	Reserved
3'b111 Reserved	4'b0000-4'b1111	Reserved

### Error Message

Error Message 用于上报 C 类错误事件给 UBFM，Correctable Error event、Non-Fatal Uncorrectable Error event、Fatal Uncorrectable Error event 的 PDU 格式均相同。

Byte 0								Byte 1								Byte 2								Byte 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	
RSVD								SCNA																								
EL	RSVD																Port_idx															

图 10-35 Error Message PDU 格式

Error Message PDU 格式如上表所示，各域段定义如下：

1. SCNA：故障发生节点的网络地址，域段长度为 24 bits；
2. Port\_idx：发生故障的端口索引，EL=0 时有效，域段长度为 16 bits；
3. EL：Error Level，域段长度为 1 bit，0 代表 Port 级错误；1 代表设备级错误；

### Link Change Message

1. Link Up/Down Message

Link Change Message 用于上报端口 Link 状态变化事件给 UBFM，触发 UBFM 处理设备动态上下线、链路故障切换和回切。Link Up event、Link Down event 的 PDU 格式相同。

Byte 0								Byte 1								Byte 2								Byte 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
RSVD								SCNA																Port_idx							
RSVD																Port_idx															

图 10-36 Link Change Message PDU 格式

Link Change Message PDU 格式如上表所示，各域段定义如下：

(1) SCNA：发生 Link 状态变化的端口的网络地址，不存在端口网络地址配置时上报 UB Controller 或 UB Switch 的网络地址，域段长度为 24 bits；

(2) Port\_idx：发生 Link 状态变化的端口索引，域段长度为 16 bits；

## 2. Hot-plug Message

Hot-plug Message 用于上报槽位工作状态，包含按键事件和在位状态变化事件。按键事件和在位状态变化事件的 PDU 格式相同。

按键消息和在位状态变化消息中各域段定义如下：

Byte 0								Byte 1								Byte 2								Byte 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
RSVD																Slot_idx															
RSVD																RSVD															

图 10-37 Hot-plug Message PDU 格式

Hot-plug Message PDU 如上表所示，各域段定义如下：

Slot\_idx：上报事件的槽位编号索引，域段长度为 16 bits。

## Configuration Message

配置消息用于配置空间寄存器读写，配置请求消息和配置响应消息由 MGMTETAH.Rsp 域段指示，承载不同的 PDU 格式，每个 Message 仅支持承载一个 Request 或 Response 操作。

Byte 0								Byte 1								Byte 2								Byte 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								
Entity_idx																RSVD								Byte_En				RSVD											
Request Address																RSVD (4 Byte)																							
Data																RSVD																							

图 10-38 配置请求消息 PDU 格式

配置请求消息 PDU 如上表所示，各字段定义如下：

1. Byte\_En: 指示是否启用本次写操作 byte，域段长度为 4 bits，分别对应 4 Bytes 数据的指示启用位。
  - 1: 启用对应 byte 写操作；
  - 0: 不启用对应 byte 写操作；
2. Entity\_idx: Entity 编号，域段长度为 16 bits；
3. Request Address: 请求地址，域段长度 4 Bytes，单位为 4 Bytes；
4. Data: 配置数据，域段长度为 4 Bytes。

Byte 0								Byte 1								Byte 2								Byte 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
Data																															
RSVD (4 Byte)																															
RSVD (4 Byte)																															
RSVD (4 Byte)																															

图 10-39 配置响应消息 PDU 格式

配置响应消息 PDU 如上表所示，各字段定义如下：

Data: 配置读响应返回数据，域段长度为 4 Bytes。

### Vendor Defined Message

1. MCTP Over UB

MCTP 消息作为配置管理消息的 PDU 在 UB 链路上传输，MCTP 消息（DMTF DSP0236）不做任何修改。

2. VDM for UB Spec

VDM for UB Spec 由本规范预留，PDU 格式如下：

Byte 0								Byte 1								Byte 2								Byte 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
UB Spec. Definition																															

图 10-40 VDM for UB Spec PDU 格式

3. VDM for Vendor

VDM for Vendor 由厂商自定义，PDU 格式如下：

Byte 0								Byte 1								Byte 2								Byte 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														
Version		Type		RSVD																Device ID																									
Vendor ID																Vendor Definition																													

图 10-41 VDM for Vendor PDU 格式

前 8 Bytes 域段来自 GUID，用于校验。

### Entity Information Exchange Message

#### 1. Resource Space Information Exchange

资源空间信息交互消息用于加速资源空间配置寄存器组的读写。资源空间信息交互请求消息 PDU 长度根据 Opcode 类型而定，当 Opcode 为 0 时，长度为 4 Bytes；当 Opcode 为 1 时，长度为 28 Bytes。响应消息 PDU 同样是变长的，当请求的 Opcode 为 0 时，长度为 36 Bytes；当请求的 Opcode 为 1 时，长度为 0。

Byte 0								Byte 1								Byte 2								Byte 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
RSVD																Opcode															
ERS0_SA0																															
ERS0_SA1																															
ERS1_SA0																															
ERS1_SA1																															
ERS2_SA0																															
ERS2_SA1																															

图 10-42 Resource Space Information Exchange 请求消息 PDU 格式

Resource Space Information Exchange 请求消息 PDU 如上表所示，各域段定义如下：

#### (1) Opcode: 操作码:

- 0: 获取资源空间信息；
- 1: 配置资源空间信息；

(1) ERS0\_SA: 资源空间 0 的起始地址，域段长度为 8 Bytes；

(2) ERS1\_SA: 资源空间 1 的起始地址，域段长度为 8 Bytes；

(3) ERS2\_SA: 资源空间 2 的起始地址，域段长度为 8 Bytes；

Byte 0	Byte 1	Byte 2	Byte 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
ERS0_SS			
ERS0_SA0			
ERS0_SA1			
ERS1_SS			
ERS1_SA0			
ERS1_SA1			
ERS2_SS			
ERS2_SA0			
ERS2_SA1			

图 10-43 Resource Space Information Exchange 响应消息 PDU 格式

Resource Space Information Exchange 响应消息 PDU 如上表所示，各域段定义如下：

- (1) ERS0\_SS：资源空间 0 的大小，域段长度为 4 Bytes，单位为 SYS\_PGS；若不存在 RS0，则该域段值为 0；
- (2) ERS1\_SS：资源空间 1 的大小，域段长度为 4 Bytes，单位为 SYS\_PGS；若不存在 RS1，则该域段值为 0；
- (3) ERS2\_SS：资源空间 2 的大小，域段长度为 4 Bytes，单位为 SYS\_PGS；若不存在 RS2，则该域段值为 0；
- (4) ERS0\_SA：资源空间 0 的起始地址，域段长度为 8 Bytes；
- (5) ERS1\_SA：资源空间 1 的起始地址，域段长度为 8 Bytes；
- (6) ERS2\_SA：资源空间 2 的起始地址，域段长度为 8 Bytes；

## 2. Obtaining Entity Information

Byte 0	Byte 1	Byte 2	Byte 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
RSVD			
MUE nums		Entity nums	
MUE0's end UE idx		MUE0's start UE idx	
MUE1's end UE idx		MUE1's start UE idx	
...			
MUEx's end UE idx		MUEx's start UE idx	

图 10-44 Entity 信息获取响应消息 PDU 格式

Entity 信息获取请求消息不携带 PDU，Entity 信息获取响应消息 PDU 如上表所示，各域段定义如下：

- (1) Entity nums: Entity 总数，域段长度为 2 Bytes；
- (2) MUE nums: MUE 数量，域段长度为 2 Bytes；
- (3) MUE[i]'s start UE idx: 第 i 个 MUE 拥有的 UE 起始索引；若该域段值等于 i，表示未绑定 UE 到该 MUE 上，域段长度为 2 Bytes；
- (4) MUE[i]'s end UE idx: 第 i 个 MUE 拥有的 UE 终止索引；若该域段值等于 i，表示未绑定 UE 到该 MUE 上，域段长度为 2 Bytes。

### 3. Link Neighbor Query

链路邻居查询允许 UBFM 查询目的节点的指定端口集合的邻居，响应消息 PDU 携带一个或多个拓扑查询响应的 Port\_info TLV，支持实现增量枚举等功能。

Byte 0								Byte 1								Byte 2								Byte 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							
D	RSVD							Port_num								RSVD																						
Port_idx0								Port_idx1																														
Port_idx2																...																						

图 10-45 链路邻居查询请求消息 PDU 格式

链路邻居查询请求消息 PDU 如上表所示，各域段定义如下：

- (1) D ( Discrete ): 指示查询模式，域段长度为 1 bit;
  - 0: 针对连续端口号的批量查询，Port\_idx0 表示起始端口号，Port\_idx1 表示结束端口号；
  - 1: 针对离散端口号的批量查询，Port\_idx0、Port\_idx1、Port\_idx2、... 分别对应要查询的端口号。
- (2) Port\_num: 本请求要查询的端口数量，域段长度为 1 Byte，UBFM 应确保响应消息的大小不超过一个管理消息的承载能力。
- (3) Port\_idx: 要查询的端口号，域段长度为 2 Bytes。

Byte 0								Byte 1								Byte 2								Byte 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	
Port_info TLVs																																

图 10-46 链路邻居查询响应消息 PDU 格式

链路邻居查询响应消息 PDU 如上表所示，各域段定义如下：

Port\_info TLVs: 响应消息返回的一个或多个 TLV，格式定义见 10.4.3.2 节拓扑查询。

### Device Security Management Message

#### 1. Device Authentication Interaction

设备认证交互消息的 PDU 格式遵守 SPDM 协议 ( DMTF DSP0274 )。

#### 2. Token Check Configuration

Token 校验配置用于启用或关闭设备端对资源空间访问请求的 Token 校验，设备端默认处于关闭状态。配置请求消息 PDU 固定为 4 Bytes，启用命令的响应消息 PDU 长度为 8 Bytes，关闭命令的响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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	
RSVD																																Check_En

图 10-47 Token 校验配置请求消息 PDU 格式

Token 校验配置请求消息 PDU 如上表所示，各域段定义如下：

Check\_En：是否启用 Token 校验，域段长度为 1 bit；0：关闭；1：启用。

Byte 0								Byte 1								Byte 2								Byte 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
RSVD																TokenID															
TokenValue																															

图 10-48 Token 校验配置响应消息 PDU 格式

Token 校验配置请求消息 PDU 如上表所示，各域段定义如下：

- (1) TokenID：访问资源空间应携带的 TokenID，域段长度为 20 bits；
- (2) TokenValue：访问资源空间携带的 TokenValue，域段长度为 32 bits；

### Pooled Resource Management Message

#### 1. Entity Registration

Entity 注册用于 UBFM 为 User 分配池化 Entity 资源，Entity 注册请求消息 PDU 长度固定为 92 Bytes，响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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															
RSVD	UPI															Entity_idx																														
EID0																																														
EID1																																														
EID2																																														
EID3																																														
GUID0																																														
GUID1																																														
GUID2																																														
GUID3																																														
RSVD								CNA																																						
UEID0																																														
UEID1																																														
UEID2																																														
UEID3																																														
ERS0_SS																																														
ERS0_SA0																																														
ERS0_SA1																																														
ERS1_SS																																														
ERS1_SA0																																														
ERS1_SA1																																														
ERS2_SS																																														
ERS2_SA0																																														
ERS2_SA1																																														

图 10-49 Entity 注册请求 PDU 格式

Entity 注册请求消息 PDU 如上表所示，各域段定义如下：

- (1) Entity\_idx: 注册给 User 使用的 Entity 索引，域段长度为 16 bits;
- (2) UPI: 注册给 User 使用的 Entity 的 UPI，域段长度为 15 bits;
- (3) EID: 注册给 User 使用的 Entity EID，域段长度为 128 bits;
- (4) GUID: 注册给 User 使用的 Entity 的 GUID，域段长度为 128 bits;
- (5) CNA: 注册给 User 使用的 Entity 的 CNA，域段长度为 24 bits;
- (6) UEID: User 的 EID，域段长度为 128 bits;

- (7) ERS0\_SS: 资源空间 0 的地址空间大小, 域段长度为 32 bits, 单位为 SYS\_PGS; 若不存在 RS0, 则该域段值为 0;
- (8) ERS0\_SA: 资源空间 0 起始地址, 域段长度为 64 bits;
- (9) ERS1\_SS: 资源空间 1 的地址空间大小, 域段长度为 32 bits, 单位为 SYS\_PGS; 若不存在 RS1, 则该域段值为 0;
- (10) ERS1\_SA: 资源空间 1 起始地址, 域段长度为 64 bits;
- (11) ERS2\_SS: 资源空间 2 的地址空间大小, 域段长度为 32 bits, 单位为 SYS\_PGS; 若不存在 RS2, 则该域段值为 0;
- (12) ERS2\_SA: 资源空间 2 起始地址, 域段长度为 64 bits;

## 2. Entity Deregistration

Entity 注销用于 UBFM 从 User 回收已分配的 Entity 资源, Entity 注销请求消息 PDU 长度固定为 20 Bytes, 响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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	
EID0																																
EID1																																
EID2																																
EID3																																
RSVD																									Reason							

图 10-50 Entity 注销请求消息 PDU 格式

Entity 注销请求消息 PDU 如上表所示, 各域段定义如下:

- (1) EID: 已分配的 Entity EID, 域段长度为 128 bits;
- (2) Reason: 注销的原因, 域段长度为 8 bits;
  - 0x00: 正常注销
  - 0x01: 设备发生异常错误
  - 0x02: 设备 link 异常
  - 0x03-0xFF: 保留

## 3. Bus Instance Create

Bus Instance 创建命令用于为虚机创建总线入口, Bus Instance 创建请求消息 PDU 固定为 36 Bytes, 响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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
GUID0																GUID1															
GUID2																GUID3															
EID0																EID1															
EID2																EID3															
RSVD																UPI															

图 10-51 Bus Instance 创建请求消息 PDU 格式

Bus Instance 创建请求消息 PDU 如上表所示，各域段定义如下：

- (1) GUID: Bus Instance 的 GUID, 域段长度为 128 bits;
- (2) EID: Bus Instance 的 EID, 域段长度为 128 bits;
- (3) UPI: Bus Instance 的 UPI, 域段长度为 15 bits;

注：若 UBPU 存在多个总线控制器，使用该命令创建的总线控制器虚拟实例会在 UBPU 所有的总线控制器上生效。

#### 4. Bus Instance Destroy

Bus Instance 销毁命令用于销毁虚机的总线入口，Bus Instance 销毁请求消息 PDU 固定为 16 Bytes，响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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
GUID0																GUID1															
GUID2																GUID3															

图 10-52 Bus Instance 销毁请求消息 PDU 格式

Bus Instance 销毁请求消息 PDU 如上表所示，各域段定义如下：

GUID: Bus Instance 的 GUID, 域段长度为 128 bits;

#### 5. Configuration Completion Notification

配置完成通知用于通知 UBPU 其总线控制器配置已完成，UB 总线驱动可以开始为上层提供服务。请求消息携带 36-Byte PDU；响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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	
RSVD																																F
GUID0																																
GUID1																																
GUID2																																
GUID3																																
EID0																																
EID1																																
EID2																																
EID3																																

图 10-53 Configuration Completion Notification 请求消息 PDU 格式

配置完成通知请求消息 PDU 如上表所示，各域段定义如下：

- (1) GUID：总线控制器虚拟实例的 GUID，域段长度为 128 bits；F 为 1 时有效；
- (2) EID：总线控制器虚拟实例的 EID，域段长度为 128 bits；F 为 1 时有效；
- (3) F：指示是否创建总线控制器虚拟实例给 UBPU 使用；0：不创建；1：创建。

注：若 UBPU 存在多个总线控制器，使用该命令创建的总线控制器虚拟实例会在 UBPU 所有的总线控制器上生效。在创建总线控制器虚拟实例时，应取总线控制器自身的 UPI 配置给总线控制器虚拟实例。

#### 6. Port Reset Notification

端口复位通知用于 UBFM 通知 UBPU 执行总线控制器端口的复位操作，包括端口复位准备和端口复位完成。UBPU 收到复位准备通知后，应执行复位前的准备工作，包括对共享总线控制器端口的其它控制器进行断流处理等。请求消息 PDU 长度固定为 4 Bytes，响应消息不携带 PDU。

Byte 0								Byte 1								Byte 2								Byte 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	
Port_idx																RSVD																T

图 10-54 Port Reset Notification 请求消息 PDU 格式

端口复位通知请求消息 PDU 如上表所示，各域段定义如下：

- (1) T：端口复位通知类型，域段长度为 1 bit；0：端口复位准备；1：端口复位完成；
- (2) Port\_idx：域段长度为 16 bits，进行复位操作的端口编号。

## 10.4.4 链路邻居通告

链路邻居通告用于一条 UB 链路连接的两个 UBPU 在建链后通过链路层 Param\_Exchg Block 交换邻居信息，包括：

1. Port Index 寄存器值
2. Entity 0 的 GUID

链路邻居通告功能支持由系统管理员根据枚举范围配置开启或关闭，UBPU 接收到链路邻居通告后，将其携带的 Port Index 和 GUID 存储到 Neighbor Port Info 寄存器组中。Neighbor Port Info 寄存器信息在端口 Link Up 后才有效，端口 Link Down 后 UBPU 应自行清空 Neighbor Port Info 寄存器信息，防止邻居节点动态上下线行为导致软件读取到错误的信息。

链路邻居通告使用的 Param\_Exchg Packet 包含 Flit0~Flit4 共 5 个 Flit，格式如下：

```
Flit0.PARAMETER_EXCHANGE_DATA.Byte0-Byte3={RSVD[15:0],port_idx[15:0]}
Flit1.PARAMETER_EXCHANGE_DATA.Byte0-Byte3={GUID[31:0]}
Flit2.PARAMETER_EXCHANGE_DATA.Byte0-Byte3={GUID[63:32]}
Flit3.PARAMETER_EXCHANGE_DATA.Byte0-Byte3={GUID[95:64]}
Flit4.PARAMETER_EXCHANGE_DATA.Byte0-Byte3={GUID[127:96]}
```

## 10.4.5 池化 Entity 管理

### 10.4.5.1 管理角色

UB Domain 内所有 Entity 声明的功能或服务组成了一个可共享的资源池，Entity 资源的管理者和使用者可以分离，UBPU 既是资源的提供者，也是资源的使用者。UBFM 是池化资源的管理者，将 Entity 动态分配给 UBPU 使用。因 Entity 0 承担了 UB Controller 的公共配置管理职责，其不应被分配给不可信的 UBPU 使用。

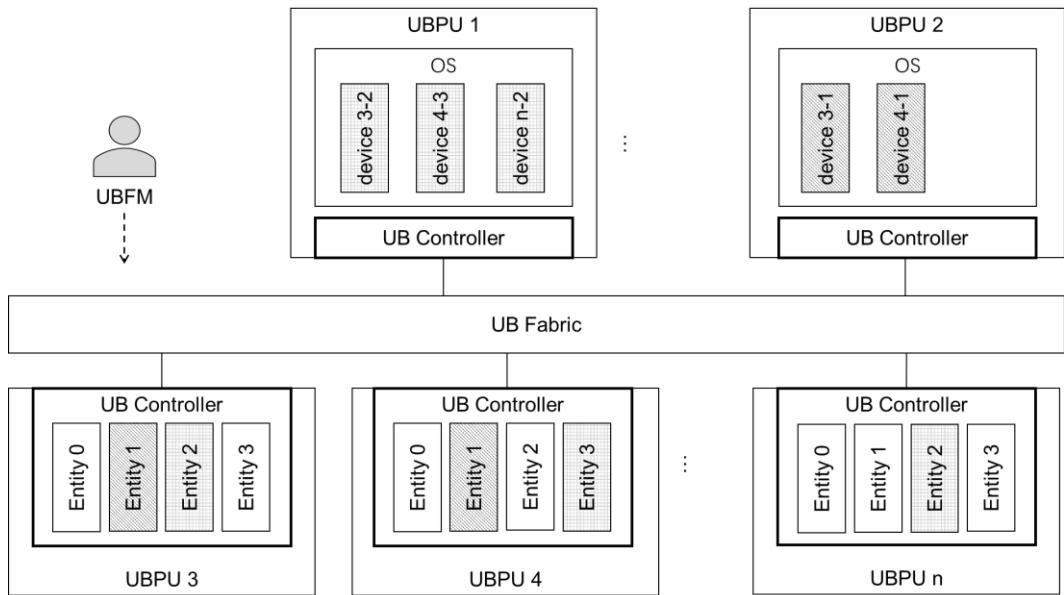


图 10-55 池化 Entity 管理示意图

#### 10.4.5.2 注册流程

UBFM 负责发现池化资源并纳入管理，包括配置网络地址和路由等。注册流程如下图所示：

1. UBFM 配置 Entity 的 EID、UPI 和 UEID。
2. UBFM 向 User 发送 Entity 注册命令，将步骤 1 中配置的 Entity 分配给 User 使用。
3. User 的 UB 总线驱动根据注册命令将 Entity 注册到 OS 中的 UB 总线后，并向 UBFM 返回命令响应。
4. 注册成功后，User 在 OS 中创建设备，并加载设备驱动，开始使用设备功能。

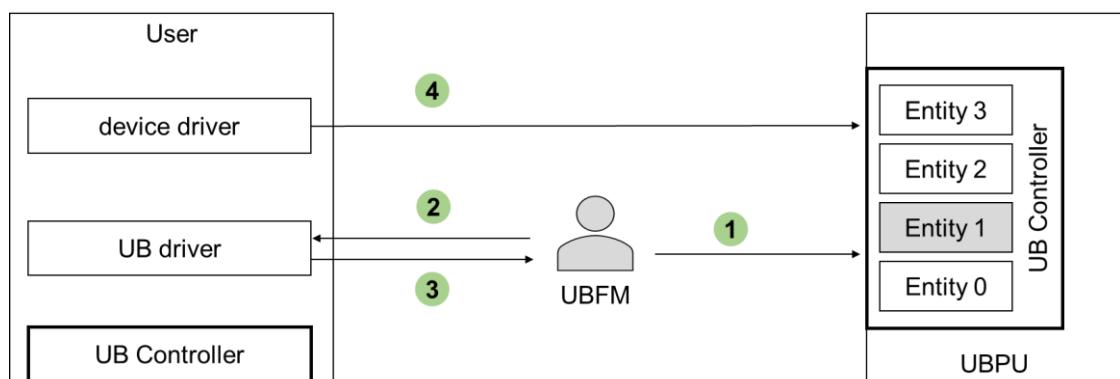


图 10-56 注册流程示意图

此过程中，UBFM 应支持：

发送 Entity Registration 命令到 User 后，应等待 User 的响应，确认注册完成。如果一段时间（由 UBFM 软件具体实现决定）内未收到 User 的响应，则认为注册失败，回收 Entity 资源。

此过程中，User 应支持：

1. 读取指定 Entity 的设备信息，确认 Entity 可用。
2. User 在收到 UBFM 的注册命令后，应记录本次注册行为，每个注册命令仅允许返回一次响应。
3. User 支持基于其本地 UB Controller（可以是总线控制器、网络控制器等）调用远端设备功能。

#### 10.4.5.3 注销流程

注销流程如下图所示：

1. UBFM 向 User 发送 Entity 注销命令，通知 User 注销已分配的 Entity。
2. User 注销设备资源，停止访问远端设备，销毁在 OS 中创建的设备。
3. User 注销完成后响应 UBFM 的 Entity 注销命令。
4. UBFM 对注销的 Entity 进行 Entity 级复位，回收 Entity 资源。

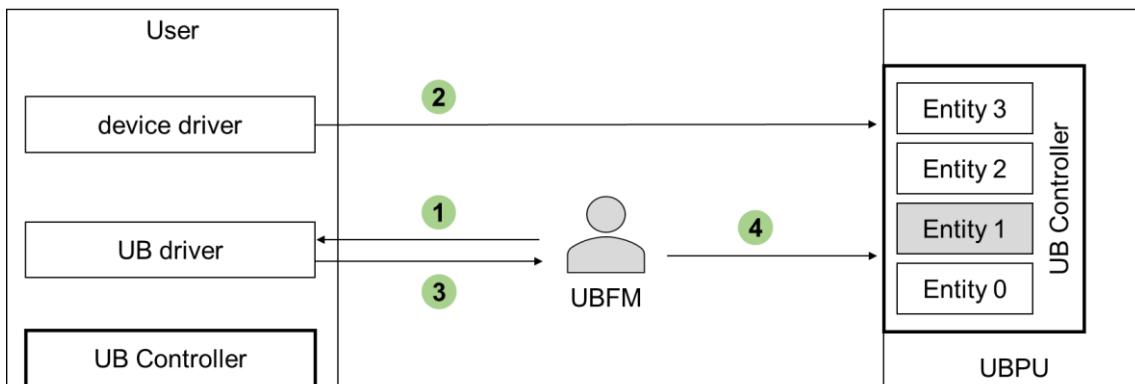


图 10-57 注销流程示意图

此过程中，UBFM 应支持：

发送 Entity Deregsistration 命令到 User 后，应等待 User 的响应，确认注销完成后再回收对应的 Entity 资源。如果一段时间（由 UBFM 软件具体实现决定）内未收到 User 的响应，则认为注销失败，强制回收对应的 Entity 资源。

此过程中，User 应支持：

1. User 在收到 UBFM 的注销命令后，应记录本次注销行为，每个注销命令仅允许返回一次响应。
2. User 在注销 Entity 时，首先注销在 OS 中为 Entity 分配的资源，确保 User 以及其授权的其它 UBPU 不会再访问该 Entity。

#### 10.4.5.4 替换流程

当分配给 User 的某个 Entity 发生不可恢复错误后，UBFM 可重新选择一个 Entity，并将故障 Entity 的基本配置迁移到新的 Entity 上，再将该 Entity 注册给 User，用于替换故障 Entity。User 侧远端设备替换的具体实现方法受 Entity 声明的设备功能约束，不在本规范定义。

### 10.4.6 通信控制

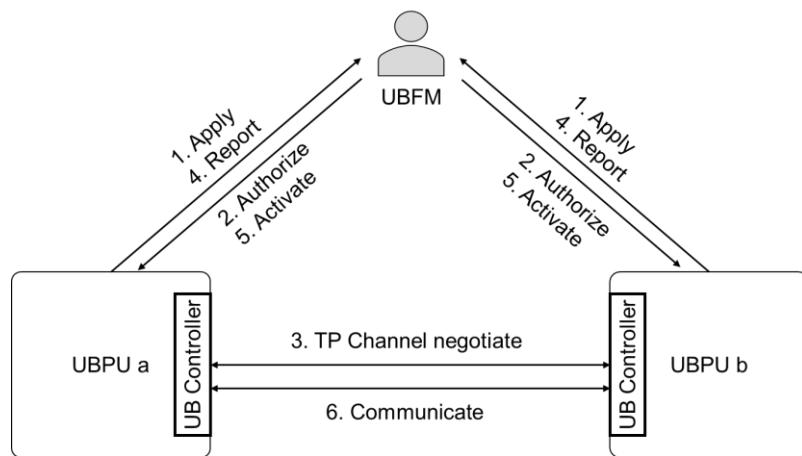


图 10-58 标准传输模式通信控制参考流程

系统管理员可按需开启 UBPU 间的通信控制，如上图所示，在使用标准传输模式时，UBPU a 和 b 在需要通信时，先向 UBFM 发起申请，UBFM 根据系统管理员预置的策略判断 UBPU a 和 b 之间是否可以通信，若允许通信，UBFM 分别向 UBPU a 和 b 进行授权。UBPU a 和 b 进行 TP Channel 协商，协商完成后上报 UBFM，UBFM 分别激活 UBPU a 和 b 对应的 TP Channel。TP Channel 被激活后 UBPU a 和 b 可以开始通信。

在使用 CTP 模式和 TP Bypass 模式时，UBFM 可以通过控制 UB Controller 路由表的表项配置实现 UBPU 间的通信控制，仅在允许通信时下发对应的路由表项。

### 10.4.7 远端内存注册控制

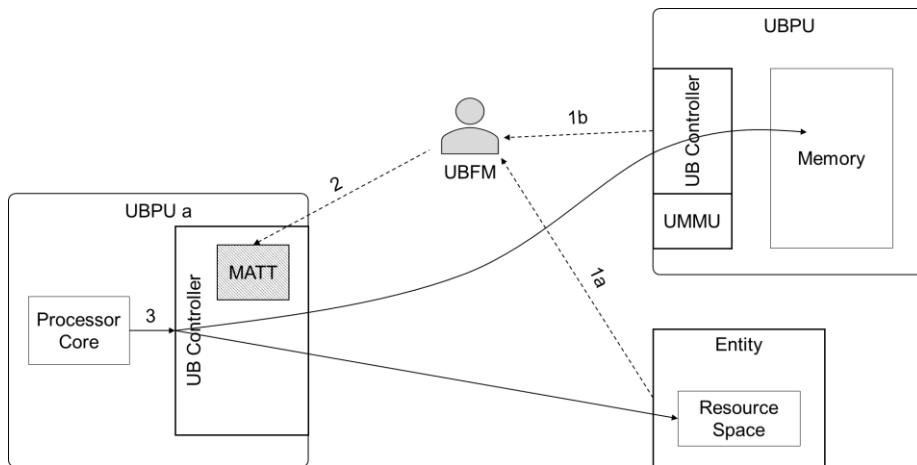


图 10-59 远端内存注册控制参考流程

系统管理员可按需开启 UBPU 的远端内存注册的集中控制，应设置 UB Controller 的 Decoder MATT jurisdiction 寄存器为 UBFM 控制模式。UBFM 在查询到 Entity 的资源空间地址信息或获取到其它 UBPU

的内存段地址信息后，将地址信息写入 UBUU a 的 Decoder 内存地址转换表，UBPU a 的处理器核即可使用 Load/Store 访问远端内存。

## 10.5 虚拟化

### 10.5.1 虚拟化机制

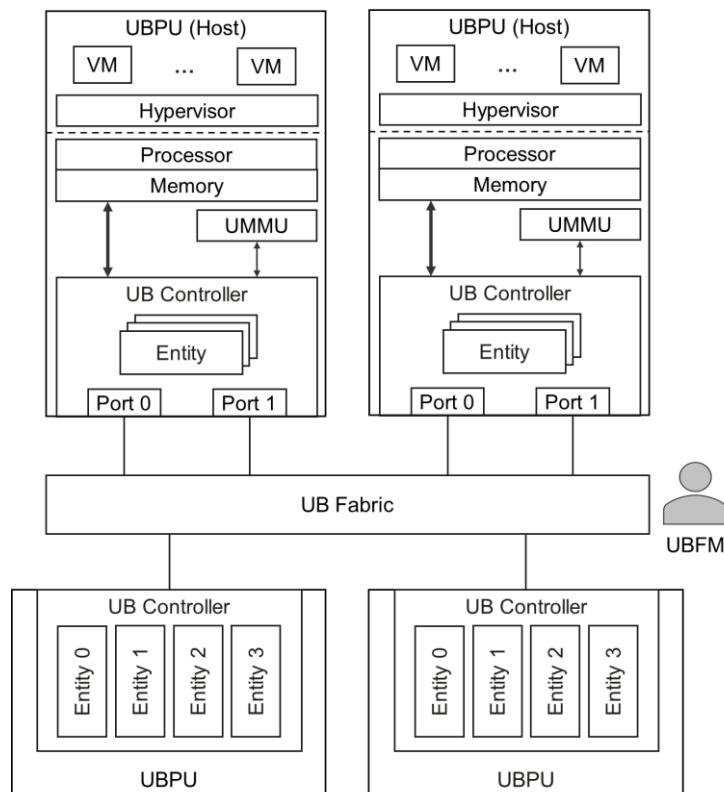


图 10-60 基于 UB 的虚拟化系统

UB 可支持硬件辅助的设备虚拟化，在 Hypervisor 虚拟化软件栈和系统平台硬件的辅助下，可以将 Entity 直通到虚机中使用。如上图所示，在虚拟化系统中，Host 包含处理器、内存、UMMU、Hypervisor、UB Controller 等软硬组件，其中 UB Controller 可以是总线控制器等。UBFM 可将池化的 Entity 分配给指定的虚机使用。已被分配的 Entity 可从虚机中卸载，经过安全处理后(如 Entity 级复位)可重新分配给其它虚机使用。

Hypervisor 虚拟化软件栈在拉起虚机时，应先为虚机创建独立的总线入口 Entity，该总线入口会被分配 EID 和 UPI，设备可以基于此总线入口 Entity 的 EID 访问虚机内存，虚机之间也可以基于此总线入口 Entity 的 EID 访问对方的内存，不同用户的虚机和设备基于 UB 分区实现访问隔离，见 10.3 节。

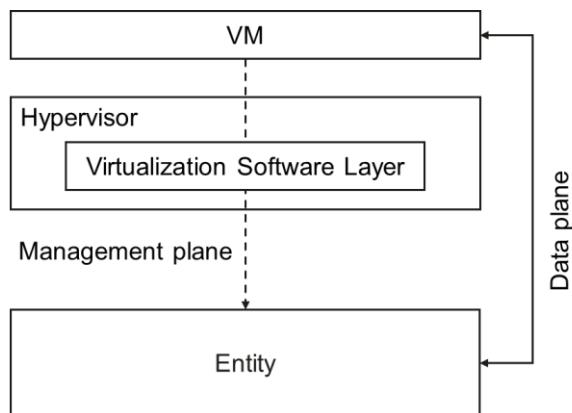


图 10-61 虚拟化机制

在 Host OS 为 Entity 分配资源后，Hypervisor 应完成如下功能以实现设备虚拟化：

1. 将 Entity 包装成一个设备给虚机使用，实现数据面直通、管理面接管。
  - (1) Entity 的数据面直通虚机，包括资源空间访问、中断、设备功能数据流，虚机的宿主机无法访问 Entity 数据面。
  - (2) Entity 的管理面由 Hypervisor 接管，对 Entity 的管理通过配置空间的访问来完成。
2. 实现对 UB 总线的支持，按照管理员指定的拓扑形式模拟互连结构，并呈现给虚机。

## 10.5.2 虚拟化管理

### 10.5.2.1 Entity 初始化

Entity 在被使用前，应通过如下流程进行初始化：

1. 配置 EID 和 UPI。
2. 配置资源空间。

### 10.5.2.2 Entity 中断

在虚拟化场景下，Entity 的中断可直通到虚机。

### 10.5.2.3 Entity 复位

在虚拟化场景下，如需要进行设备的替换，可以对被替换设备进行 Entity 级复位，确保信息安全，并让 Entity 回到初始状态；新的 Entity 导入虚机之前，可以进行 Entity 级复位，使 Entity 处于初始状态。

## 10.6 RAS

### 10.6.1 复位

#### 10.6.1.1 一般要求

UBPU 应支持复位功能，包括设备级复位、Entity 级复位、Port 级复位。

UBPU 应支持基于管理命令下发的复位操作，可支持 Pin 信号触发的复位操作，也可支持自复位，自复位操作的具体实现不在本规范定义。

#### 10.6.1.2 设备级复位

##### 10.6.1.2.1 复位范围

设备级复位将一个 UBPU 内所有 Entity 及其声明的功能或服务、Port 全部复位到上电初始状态。设备级复位过程中，可能会导致未完成的任务异常或响应超时。

##### 10.6.1.2.2 管理命令复位

设备级复位寄存器只在 Entity 0 中存在，其它 Entity 不存在该寄存器。

UBPU 收到设备级复位命令后立即执行复位操作，无需回复响应。UBFM 可延迟查询复位完成状态。

##### 10.6.1.2.3 Pin 复位

UBPU 可支持通过信号线触发的设备级复位操作，当该信号线的电平满足要求时，触发复位操作。

#### 10.6.1.3 Entity 级复位

Entity 级复位将 Entity 的配置空间 1 和资源空间复位到上电初始状态，Entity 级复位不影响网络层、数据链路层、物理层等公共资源的配置。

Entity 的 User 可通过配置管理命令控制各自使用的 Entity 进行复位。

注：在接收到 Entity 级复位命令后，Entity 宜尽快完成复位操作，且保证复位后可以被正常访问。

#### 10.6.1.4 Port 级复位

Port 级复位将指定 Port 的配置复位到上电初始状态，复位后应对 Port 进行重新配置。UBFM 通过配置管理命令写 Port 的复位寄存器可以实现 Port 级复位。

Port 在复位状态时，应在路由表中将其描述为不可使用状态。当一个 UBPU 的所有 Port 都处于复位状态时，该 UBPU 将无法被访问。

## 10.6.2 错误处理

### 10.6.2.1 一般要求

UB 定义了一系列的错误检测、处理、记录和上报的机制，支持故障定位和恢复。错误处理应遵守如下原则：

1. 不扩散错误，降低错误影响范围；
2. 支持快速恢复，缩短错误影响时间；
3. 分类处理和就近处理策略；
4. 提供错误记录。

### 10.6.2.2 错误类型

#### 10.6.2.2.1 错误分类

本规范将错误事件分为以下三个类别：

1. A 类：此类错误的特征是 UB Fabric 的物理拓扑是完好的，错误发生之后能对应到单个事务，错误发生之后不影响其它事务的执行。例如 SQ 中的单个 SQE 执行发生了错误，通常可以通过 CQ 上送错误事件，由软件进行处理。
2. B 类：此类错误的特征是 UB Fabric 的物理拓扑是完好的，但错误发生之后无法对应到单个事务，仅能对应到单个 Entity、TP Channel 或 Jetty 等（Entity 级错误、TP Channel 级错误、Jetty 级错误等），需要对错误单元的全部资源进行处理。通常可以上报错误事件，由设备驱动进行处理。
3. C 类：此类错误的特征是设备级或端口级的公共资源发生错误。例如 UB Fabric 的物理拓扑损坏，通常可以上报错误事件，由 UBFM 进行处理。C 类错误可以进一步分为可纠正错误（Correctable Error）和不可纠正错误（Uncorrectable Error）两类：
  - (1) 发生可纠正错误后可以自恢复，业务不会受到影响。
  - (2) 发生不可纠正错误后无法自恢复，需要软件介入处理。不可纠正错误可进一步细分为 Non-Fatal Uncorrectable Error (NFUE) 和 Fatal Uncorrectable Error (FUE)，其中 FUE 造成的影响更大。不可纠正错误中的部分错误（具体错误详见附录 D）可以通过配置寄存器设置为 NFUE 或 FUE。

设备应提供自下而上的分层错误检测能力，并将检测到的错误对应到本规范定义的错误类别。

### 10.6.2.2 A 类错误列表

表 10-12 A 类错误列表

错误编号	错误类型	错误含义	错误类别及行为
A.1	Unsupported Opcode	SQE 中的 Opcode 为非法值。	A 类：通过 CQE 上报错误。Status 为 0x01，Sub-Status=0x0。 若无法通过 CQE 上报错误时，可退化为上报 B 类 JFS Check Error。
A.2.1	Local Length Error	SQE 或 RQE 的 Length 为非法值。	A 类：通过 CQE 上报错误，Status 为 0x02，Sub-Status=0x1。 若无法通过 CQE 上报错误时，可退化为上报 B 类 JFS Check Error 或 JFR Check Error。
A.2.2	Local Access Error	本端的内存访问发生权限校验错误。	A 类：通过 CQE 上报错误，Status 为 0x02，Sub-Status=0x2。
A.2.3	Remote Response Length Error	Initiator 收到的 Response 存在长度错误。	A 类：通过 CQE 上报错误，Status 为 0x02，Sub-Status=0x3。
A.2.4	Local Data Poison	读取本端数据时发生内存 Poison 错误或接收到的 Poison 数据需要写入本端内存。	A 类：通过 CQE 上报错误，Status 为 0x02，Sub-Status=0x4。
A.3.1	Remote Unsupported Request	Initiator 收到 Target 返回远端操作请求的检查错误。 TAACK.RSPST=Completer Error(0x3), RSPINFO=Unsupported Request(0x1)。	A 类：通过 CQE 上报错误，Status 为 0x03，Sub-Status=0x1。
A.3.2	Remote Access Abort	Initiator 收到 Target 返回远端操作请求的执行错误。 TAACK.RSPST=Completer Error(0x3), RSPINFO= Remote Abort(0x2)。	A 类：通过 CQE 上报错误，Status 为 0x03，Sub-Status=0x2。
A.3.3	Remote Data Poison	Initiator 在读取远端数据时，在 Target 发生 Poison 错误。	A 类：通过 CQE 上报错误，Status 为 0x03，Sub-Status=0x4。
A.4	Transaction Retry Counter Exceeded	事务层 RNR 重传超过门限值。	A 类：通过 CQE 上报错误，Status 为 0x04，Sub-Status=0x0。
A.5	Transaction Ack Timeout	事务应答超时。	A 类：通过 CQE 上报错误，Status 为 0x05，Sub-Status=0x0。

### 10.6.2.2.3 B 类错误列表

表 10-13 B 类错误列表

错误编号	错误类型	错误含义	错误类别及行为
B.1.1	JFS Check Error	JFS 校验错误。	B 类: 上报 Jetty 级错误异步事件, Event_Type=0x1, Sub_Type=0x1。
B.1.2	RSVD	Reserved	-
B.1.3	JFC Check Error	1. JFC 校验错误。 2. JFC 发生 Overflow。	B 类: 上报 Jetty 级错误异步事件, Event_Type=0x1, Sub_Type=0x3。
B.1.4	RSVD	Reserved	-
B.2.1	Transport Retry Counter Exceeded Error	传输层重传超次。	B 类: 上报 TP 级错误异步事件。Event_Type=0x2, Sub_Type=0x1。
B.2.2	Transport Unexpected Opcode	TP Receiver 接收的数据包 PSN 正确, 但 TP Opcode 不符合预期。直接丢弃。	B 类: 上报 TP 级错误异步事件。Event_Type=0x2, Sub_Type=0x2。
B.2.3	Transport PSN and Transaction MSN Mismatch	PSN 与 MSN 不匹配。	B 类: 上报 TP 级错误异步事件。Event_Type=0x2, Sub_Type=0x3。
B.3	Entity Level Error	发生只能对应到 Entity 的错误。	B 类: 上报 Entity 级错误异步事件。Event_Type=0x3, Sub_Type=0x0。

### 10.6.2.2.4 C 类错误列表

C 类错误根据影响范围可区分为 Port 级和设备级。当错误无法确定为某个 Port 时, 上报设备级错误, Error Message 的 EL 域设置 1; 当错误能确定为某个 Port 时, 上报 Port 级错误, Error Message 的 EL 域设置 0。

C 类错误主要出现在物理层、数据链路层和网络层, 物理层和数据链路层的所有错误均为 Port 级, 错误记录到 Uncorrectable Error Status 或 Correctable Error Status 寄存器。

表 10-14 C 类错误列表

错误编号	错误类型	错误含义	错误类别及行为
C.1	RSVD	Reserved	-
C.2	Block Align Unlock	Block 失去锁定。	默认上报 CE 错误, 通过 Error

错误编号	错误类型	错误含义	错误类别及行为
			Message 上报 Port 级错误。
C.3	Elasticity Buffer Overflow/ Underflow	弹性 Buffer 上溢出或下溢出。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.4	Loss of Lane-to-Lane Deskew	Lane 间的 Deskew 失锁。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.5	AMCTL decode Error	AMCTL 解码错误。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.6	LTB CRC Error	LTB CRC 校验错误。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.7	Link speed reduce Error	出现非预期的降速事件。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.8	Link width reduce Error	出现非预期的降 Lane 事件。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.9	Equalization Coarsetune Timeout	Equalization 粗调阶段超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.10	Equalization Finetune Timeout	EQ.Passive 状态或 EQ.Active 状态超时	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.11	Link width negotiation Timeout	Link width 协商超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.12	LMSM Discovery State Timeout	LMSM Discovery 阶段超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.13	RSVD	Reserved	-
C.14	RSVD	Reserved	-
C.15	QDLWS-timeout	动态升降 Lane 超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.16	Unsupported LW Switch REQ	收到不支持的 Link width 切换请求。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.17	RSVD	Reserved	-
C.18	LMSM Retrain state Timeout	LMSM Retrain state 超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.19	DL Retry ACK Timeout	DL 发送端等待 Retry_Ack_Set 超时。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。

错误编号	错误类型	错误含义	错误类别及行为
C.20	DL Retry Rollover	DL 重传相同 RcvPtr 的 Packet 超次。	默认上报 CE 错误，通过 Error Message 上报 Port 级错误。
C.21	DL Protocol Error	数据链路层协议错误，包括： 1. 信用证返还超时； 2. 返还的 ACK 导致 Retry Buffer 溢出。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.22	DL Retry Error	DL 层重传状态机进入 Error 状态。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.23	RSVD	Reserved	-
C.24	Packet Length Error	数据包的长度检查出错： 1. 接收端收到超长包，即数据包总长 (包含 Header+Payload) 超过协议定义的最大值，或数据包的 Payload 长度超过传输层 MTU。 2. 接收端接收到超短包，即数据包总长 (包含 Header+Payload) 小于协议理论的最小值。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.25	RSVD	Reserved	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.26	RSVD	Reserved	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.27	RSVD	Reserved	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.28	ICRC Check Error	接收方向 ICRC 校验错误。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.29	Receive Buffer Overflow	接收端 Receive Buffer 溢出错误。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。
C.30	Flow Control Overflow	发送端接收 CRD 后超出初始化信用证，流控溢出错误。	默认上报 NFUE 错误，可通过 Error Message 上报 Port 级错误。

### 10.6.2.3 错误处理策略

#### 10.6.2.3.1 A 类错误处理策略

A 类错误通过 Completion Queue 上报时，错误通常上报至对应的编程接口进行处理。因为 A 类错误可以对应到某个事务，用户可以根据实际应用场景采用以下处理策略：

1. 软件层面进行事务重试；
2. 复位 Jetty；
3. 复位 TP Channel；
4. 复位 Entity。

#### 10.6.2.3.2 B 类错误处理策略

B 类错误通过 Event Queue 上报时，通常交由对应的设备驱动进行处理。设备驱动可根据错误码采用以下处理策略：

1. 复位 Jetty；
2. 复位 TP Channel；
3. 复位 Entity。

#### 10.6.2.3.3 C 类错误处理策略

C 类错误通过 Error Message 上报时，通常先上报给本地 Firmware 处理，Firmware 进行预处理后再将错误上报给 UBFM 处理。UBFM 可根据错误的严重级别进行 Port 级、设备级复位。

### 10.6.2.4 错误记录

#### 10.6.2.4.1 一般要求

错误记录功能是指硬件将检测到的错误按指定格式记录到寄存器，后续软件的错误搜集定位及错误恢复都依赖错误记录。错误记录主要基于下述原则进行设计：

1. 错误记录需要提供尽可能完备的错误信息，支持快速寻找到出错点并精准地进行错误恢复。
2. 错误记录功能定义统一且标准的记录格式。

使用 Event Queue 和 Completion Queue 上报的错误在事务层的相关数据格式中说明，下面的内容主要体现以下两种错误记录方式：

1. 错误现场记录 Error Record。
2. 错误消息队列 Error Message Queue。

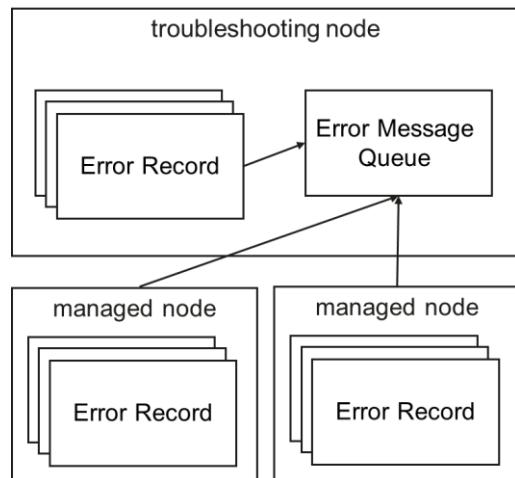


图 10-62 错误记录

#### 10.6.2.4.2 错误现场记录

错误现场记录用于存储错误产生的第一现场的错误信息，详细的错误现场记录寄存器请参照配置空间的 CFG0\_CAP 和 CFG0\_PORT\_CAP 中的 ERR\_RECORD 寄存器组（见附录 D）。

#### 10.6.2.4.3 错误消息队列

错误消息队列（Error Message Queue）通常只位于部署了错误信息处理软件的节点，即错误上报的目的节点。错误消息队列由软件进行配置，详细的错误消息队列寄存器见 D.2.4 节。

# 11 安全

## 11.1 概述

### 11.1.1 信任模型

#### 11.1.1.1 资产保护目标

UB 的安全目标在于保护通过 UB 协议栈互访的数据资产安全，包括但不限于：

- UBPU 的设备身份、固件和配套软件。
- 内存数据。
- 总线传输数据。
- 各安全功能涉及的密钥、访问凭据和配置参数等敏感数据。

#### 11.1.1.2 安全假设

假设工作在有安全保障措施的机房中的 UBPU 不会遭受物理攻击，包括但不限于硬件插拔、故障注入等。

假设 UBFM 是可信的，负责设备安装部署、运维管理的人员都是经过身份鉴别与权限校验的，实施的管理操作也是可靠的，不会滥用权限恶意配置错误参数等。

通过拒绝服务攻击、资源耗尽型攻击和劫持控制 UB Switch 等破坏协议工作流程，影响可用性的攻击不在本文件范围内。

本文件聚焦于基于 UB 的设备互联互通安全，UBFM、UBPU 和 UB Switch 本身的安全启动、硬件可信根、访问控制等安全机制、设备和协议本身的软硬件设计与实现缺陷不在本文件范围内。

#### 11.1.1.3 可信计算基

UB Domain 中需要用户初始信任的可信计算基（ Trusted Computing Base, TCB ）包括：UBPU 的硬件可信根，硬件安全模块（ Hardware Security Module, HSM ）和硬件可信执行环境（ Trusted Execution Environment, TEE ）相关的计算单元及其中预置的随机数、根密钥和根证书等。同时，在 UB Domain 运行中，用户可根据需求通过可信启动等机制进一步扩展 TCB 。

## 11.1.2 安全威胁

UB 安全能够抵御以下安全威胁：

- **UBPU 的身份仿冒、固件篡改或替换：**攻击者可能使用不可信的 UBPU 物理仿冒、替换受信任的 UBPU，或篡改受信任的 UBPU 固件等，进一步在互联通信中攻击其他 UBPU。
- **未经权限验证的内存或 Jetty 数据访问：**未经权限验证地访问内存或 Jetty 中的数据。
- **数据包窃听、篡改、注入、重放或伪造：**跨信任域的 Entity 通信时，攻击者可通过监听通信链路、控制其中的交换机、路由器或网络线路，以及通过侧信道攻击等，对 UBPU 之间传输的数据包进行非法窃听、篡改、注入、重放或者仿冒数据包源身份进而窃听和伪造通信数据。
- **对设备可信执行环境内部和通信数据的非法访问：**可信执行环境之外的进程可能对设备可信执行环境内或可信执行环境之间通信数据进行非法访问。

## 11.1.3 功能简介

UB 提供以下安全功能：

- **设备认证：**对 UBPU 进行可信度量与安全认证。
- **资源分区隔离：**分别从网络层和事务层配置 NPI 与 UPI，实现 UB 资源分区隔离。
- **访问控制：**分别对通过 UB 互联的内存和 Jetty 进行访问权限校验，防范非授权访问。
- **数据通路保护：**对基于 UB 协议栈通信的数据流进行加解密和完整性校验，保护数据包的机密性和完整性。
- **可信执行环境扩展：**提供跨设备的可信执行环境安全扩展机制。

安全功能与安全威胁的映射关系见表 11-1，用户可根据各应用场景，对安全风险的接受程度，灵活选择部署相应的安全功能：

表 11-1 安全功能与安全威胁的映射

安全功能	削减的安全威胁
设备认证	UBPU 的篡改或替换
资源分区隔离	未经权限验证的内存或 Jetty 数据访问
访问控制	未经权限验证的内存或 Jetty 数据访问
数据通路保护	数据包窃听、篡改、注入、重放或伪造
可信执行环境扩展	对 UBPU 的 TEE 内部和 TEE 之间通信数据的非法访问

## 11.2 设备认证

### 11.2.1 应用场景

UB 提供设备认证功能，包括设备接入认证和设备间互联认证，认证内容可包含 UBPU 的身份、软硬件组件的完整性度量值等。

设备认证功能的应用场景包括但不限于：

- 远程系统管理员可以通过主板管理控制单元 ( Baseboard Management Controller, BMC ) 或其它平台可信根，在无需对系统进行物理检查的条件下，对 UBPU 进行认证。
- 在系统运行时/热插拔场景中，在为 UBPU 的 Entity 分配资源之前，UBFM 或系统管理员需要验证或主册 UBPU 的身份和配套固件的完整性。
- 不同 UBPU 中的 Entity 在通信之前，可以进行双向身份认证。

### 11.2.2 接入认证

设备接入认证是指在 UBPU 接入 UB Domain 时，UBFM 需要先对 UBPU 的身份和可信状态进行认证，认证通过后允许 UBPU 接入。UB 协议栈支持三种认证流程，第一种是 UBFM 直接认证 UBPU 身份，第二种是 UBFM 基于 UBPU 的度量启动能力并协同验证服务器（包括本地或远程证明服务器），认证设备的身份和可信状态，第三种是由管理员确定设备身份和可信状态后，向 UBFM 注册设备信息。

UBFM 直接认证 UBPU 身份，即 UBFM 向 UBPU 发送身份认证挑战请求，获取 UBPU 的数字证书，通过验证数字证书确认 UBPU 的身份。

在基于度量启动的认证方式中，UBFM 认证的内容可包含 UBPU 的身份和固件与配套软件的度量值等，认证过程包括 UBFM 发起认证请求，获取 UBPU 的完整性度量报告、UBPU 身份证书链、度量报告签名等，UBFM 通过本地或者远程证明机制验证度量报告中的度量值，以认证 UBPU，并为认证通过的 UBPU 颁发经过 UBFM 签名的认证凭据。当 UBPU 退出 UB Domain 时，UBFM 注销其认证凭据。

此外，考虑到 UB Domain 内可能存在大量具有相同固件和配套软件的同质化 UBPU，UBFM 对设备的认证过程可以做简化和加速，认证过程见图 11-1，核心步骤如下：

1. UBPU 在启动过程中，生成固件和配套软件的度量值，见图 11-1 的步骤 1。
2. UBFM 向 UBPU 发起认证挑战请求，获取 UBPU 的度量报告摘要以及 UBPU 对摘要和挑战随机数的数字签名，见图 11-1 的步骤 2、3。
3. UBFM 验证 UBPU 度量报告摘要的数字签名后，判断该摘要所对应的度量值是否与已验证过的 UBPU 的度量值相同，如果 UBPU 发送的度量报告摘要与已经验证过的度量报告不相同，则需获取 UBPU 的完整度量报告，并提交验证服务器对比验证，验证通过则为 UBPU 颁发经过 UBFM 签名的认证凭据，见图 11-1 的步骤 4.1、5、6、7、8 和 9.1。

4. 如果 UBFM 判断 UBPU 发送的度量报告摘要与已验证过的度量报告相同，则可以跳过获取完整度量报告和提交验证服务器对比验证过程，直接为 UBPU 颁发经过 UBFM 签名的认证凭据，见图 11-1 的步骤 4.2 和 9.2。
5. UBFM 还可以生成集群度量报告（多个 UBFM 实例可分别为其管理的子集群生成度量报告），包括但不限于集群中的 UBPU 列表和每一种独特 UBPU 对应的度量报告等，以供用户或管理员对集群中 UBPU 整体进行快速验证。

注：图 11-1 中步骤 2、6、7 和 8 可参考 DSP0274 SPDM 规范。

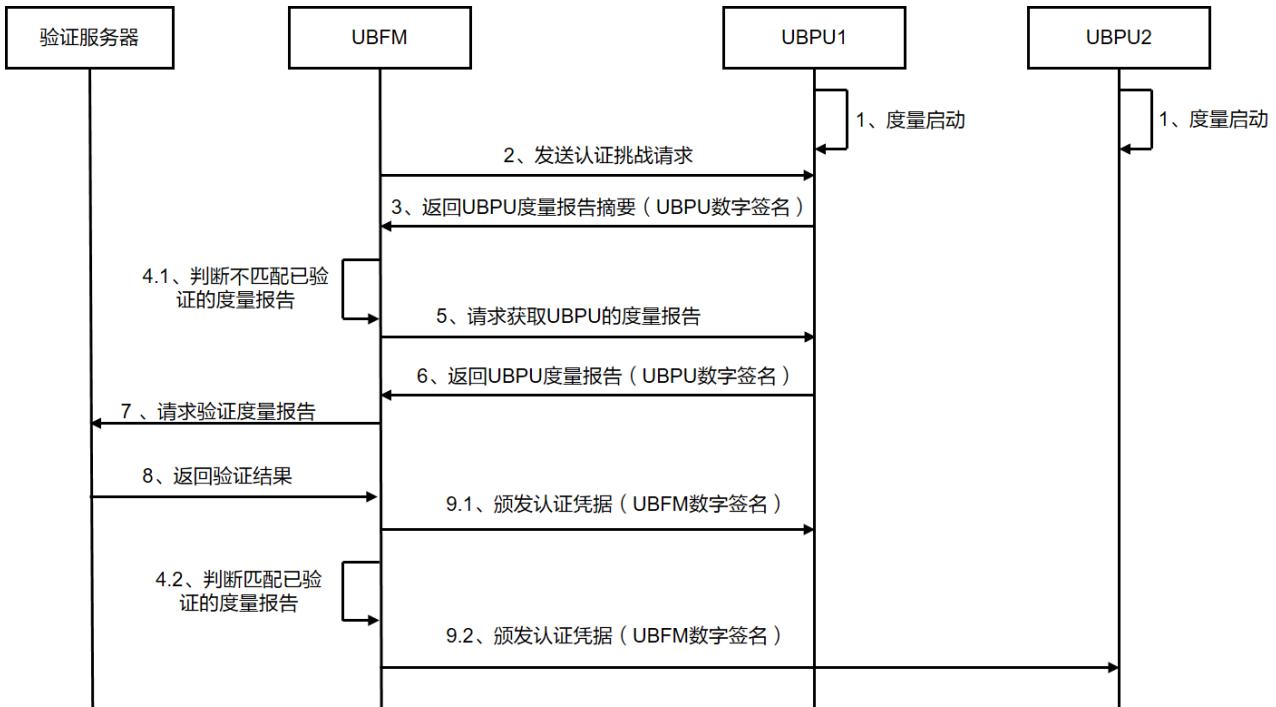


图 11-1 设备接入认证

考虑到部分 UBPU 可能不具备度量启动能力，UB 允许由管理员确定 UBPU 的身份和配套固件完整性后，在 UBFM 中注册设备信息，允许其加入 UB Domain。由 UBFM 为 UBPU 颁发经过 UBFM 数字签名的设备认证凭据。当 UBPU 退出 UB Domain 或认证凭据失效时，UBFM 注销其认证凭据。

### 11.2.3 互聯認證

设备间互聯認證是指 UBPU 之间通信时，设备间进行双向身份认证。设备间互聯認證可基于数字证书等设备身份进行认证，可选增加对设备的完整性度量报告等进行认证，还可以通过验证由 UBFM 为 UBPU 颁发的经过数字签名的认证凭据，验证设备身份和可信状态。

## 11.3 资源分区隔离

### 11.3.1 Entity 分区隔离

UB 协议栈通过 UB 分区实现 Entity 分区隔离，UPI 标识了一组 Entity 的集合，在 UB Domain 内具有唯一性，用于实现 Entity 粒度的资源隔离，能够帮助用户实现不同业务功能的安全隔离。UPI 的定义见 10.3.2 节，UPI 对应的包格式见附录 B.3。

UPI 由 UBFM 管理并配置，不接受其它组件修改配置 UPI 值，UBFM 会对 UPI 及 Entity 所属分区的映射关系进行安全管理，确保 UPI 能够正确地下发到合法的 Entity。发送侧基于 UBFM 分配的 UPI 值组装数据包的 UPI 域段，接收侧实现 UB 协议栈的硬件解析出数据包中的 UPI，并与 UBFM 配置的 UPI 进行对比校验，校验一致，才处理和响应该数据包。

注：上述过程适用于 Entity 之间的数据通信，UBFM 发出的数据包不受 UPI 校验约束。

### 11.3.2 网络分区隔离

UB 协议栈通过 NPI 实现网络分区隔离，协议在网络层定义 NPI，NPI 在 UB Domain 内具有唯一性。NPI 支持可信配置，可用于物理机用户隔离、网络平面隔离和交换机控制面访问隔离等多种场景。NPI 的域段定义和校验机制见 5.3.3.1 节。

## 11.4 访问控制

### 11.4.1 应用场景

UB 提供了事务层的访问控制功能，访问控制功能的应用场景见表 11-2，包括内存访问和 Jetty 访问两个场景。内存访问场景的访问控制功能实现依赖于 UMMU 的权限控制表，见 9.4.4 节，UMMU 权限控制表与地址翻译表独立，内存访问时权限校验与地址翻译分别处理，当两者均操作成功则允许内存访问，否则拒绝内存访问。

表 11-2 访问控制功能应用场景

应用场景	是否引入 TokenValue	访问凭据标识	是否引入 UMMU 协助卡控
内存访问	可选	TokenID	是
Jetty 访问	可选	TCID	否

## 11.4.2 功能原理

以内存访问为例，UB 的访问控制功能原理如下：

1. User 申请访问 Home 特定内存段时，Home 验证 User 身份后，向 User 返回 TokenID 和一个随机数作为 TokenValue。
2. User 访问 Home 内存时，在数据包中携带 TokenID 和 TokenValue，Home 查表比对内存地址和 TokenValue 等，校验成功后允许访问。

注 1：TokenValue 随机数可以通过读取 UBU 上的真随机数发生器获得；也可以由伪随机数发生器产生伪随机数。用户可根据业务场景的安全威胁分析，判断是否采用密码技术保护 TokenValue 的机密性和完整性。

注 2：在一定的时间窗内，Home 发生虚拟机重启或应用进程重启等情况，可能导致过时的数据包访问目的资源引发数据错误，应采取相应措施，避免此类情况发生，例如，在 Home 重启后，Home 立即更新 TokenValue。

注 3：Home 可能为多个访问相同内存段并具有相同操作权限的 User，返回相同的 TokenID 和 TokenValue，也允许多个 User 之间共享 TokenID、TokenValue 和内存段等信息，共享方式包括但不限于一个 User 申请获得内存段的访问权限后，将内存段和 TokenID 广播给其他 User，其他 User 如需获得访问权限，则进一步向已获得访问权限的 User 申请获得 TokenValue 信息（可加密传输）。

内存访问场景，UMMU 在基于 TokenValue 访问权限管控的基础上，提供 E\_Bit，User 能否访问 Home 内存段还需受 E\_Bit 校验约束，E\_Bit 校验规则见 9.4.4.3.3 节。

UB Domain 边界，需对基于 IP 地址的数据包进行过滤，可配置仅允许 E\_Bit 为 0 且携带 TokenValue 的数据包进入。

## 11.4.3 权限分配流程

UB 访问控制功能在内存访问场景的核心工作流程如下：

1. User 向 Home 申请访问权限时，由 Home 验证 User 身份成功后，Home 为 User 注册申请相关资源，获得 TokenID、内存段 Segment、操作权限集合 Segment\_permission 及其 TokenValue（可选），并将 TokenID、内存段 Segment、操作权限集合 Segment\_permission 及其 TokenValue（可选）安全传回 User。
2. User 发起对 Home 的内存访问时，User 在数据包中携带 TokenID、内存地址、操作类型信息及 TokenValue（可选）域段，Home 侧依赖 UMMU 校验对应 TokenID 是否存在对应表项，并且对比数据包中的 TokenValue 域段与页表项中的存储的 TokenValue 是否相同，基于比对结果判断 User 能否访问目标内存地址。

UB 访问控制功能在 Jetty 访问场景的核心工作流程如下：

1. Initiator 在访问 Target 之前，向 Target 申请 TCID 和 TokenValue，Target 生成 TCID 和 TokenValue 后，将 TCID 和 TokenValue 安全地传回 Initiator。

2. Initiator 在发送给 Target 的数据包中携带 TCID 和 TokenValue，Target 收到数据包后，通过数据包中的 TCID 来索引 TokenValue，并将数据包中的 TokenValue 与 Target 的 TokenValue 进行对比，从而判断 Initiator 是否有访问权限。

不同业务场景，用户使用访问控制功能时，可根据需求配置不同的策略，具体如下：

- 只传输 TokenID 或 TCID，不带 TokenValue，这种方式性能最高，但是存在非授权访问风险。
- 传输 TokenID 或 TCID 和 TokenValue 明文，这种方式具有一定安全性，但是存在网络中间节点非法获取 TokenValue 的风险。
- 传输 TokenID 或 TCID 和 TokenValue，并加密保护，而 PLD 是明文，这种方式网络中间节点无法获取 TokenValue，但是可看到 PLD 内容。
- 传输 TokenID 或 TCID 和 TokenValue，并加密保护，PLD 也加密，这种方式最安全，但是带来的硬件开销较大。

#### 11.4.4 权限无效化流程

UB 访问控制功能存在两种粒度的访问权限无效化机制：

- 权限组粒度无效化：

权限组粒度无效化，是由 Home 发起，Home 侧 UMMU 相关软硬件具体执行，无效化 Home 侧 UMMU 中的 TokenID 和对应的 TokenValue。此时，持有该 TokenID 和 TokenValue 的权限组内所有 User 访问权限均被无效化。

- 用户粒度无效化：

用户粒度无效化是指无效化权限组内具体 User 的访问权限，同时保证最小化影响权限持续有效的 User，是由 User 发起。步骤如下：

- (1) 多个 User 均申请获得 Home 的内存访问凭据（可以是分别向 Home 申请，也可以是多个 User 之间共享获得），包括 TokenID 和相应的 TokenValue。
- (2) Home 更新 TokenValue，向权限持续有效的 User 分发更新后的 TokenValue，权限失效的 User 将不会收到更新后的 TokenValue。
- (3) Home 仅接受基于更新后的 TokenValue 申请内存访问，从而实现无效化未收到更新后的 TokenValue 的 User 的内存访问权限。

注：Jetty 访问场景，权限无效化流程与上述过程类似。

示例：

权限组内，对 User1 的用户粒度权限无效化示例见图 11-2。具体步骤如下：

1. 时间点 1：User1、User2、User3 均申请获得 Home1 的 TokenID1 和主访问凭据 A-TokenValue，后续 User1、User2、User3 通过 TokenID1、A-TokenValue 访问 Home1 对应内存，Home1 同时维护备访问凭据 B-TokenValue。
2. 时间点 2：Home1 向 User2、User3 发送 B-TokenValue，User2、User3 基于 B-TokenValue 作为访问凭据；Home1 未向 User1 发送 B-TokenValue；过渡阶段（具体过渡时长由 Home1 决定），Home1 权限页表同时支持 A-TokenValue

和 B-TokenValue 的校验，校验成功则允许访问内存；对于高安全业务场景，Home1 也可以直接将主访问凭据即 A-TokenValue 更新为 B-Tokenvalue，仅向 User2 和 User3 发送 B-TokenValue，并将备访问凭据更新为 C-Tokenvalue，使得 User1 立即失去对 Home1 的访问权限。

- 时间点 3：经过过渡阶段后，Home1 将主访问凭据 A-TokenValue 替换为 B-TokenValue，同时生成 C-TokenValue 作为新的备访问凭据；此时，任何基于 A-TokenValue 的访问权限校验失败，从而完成对 User1 的访问权限无效化。

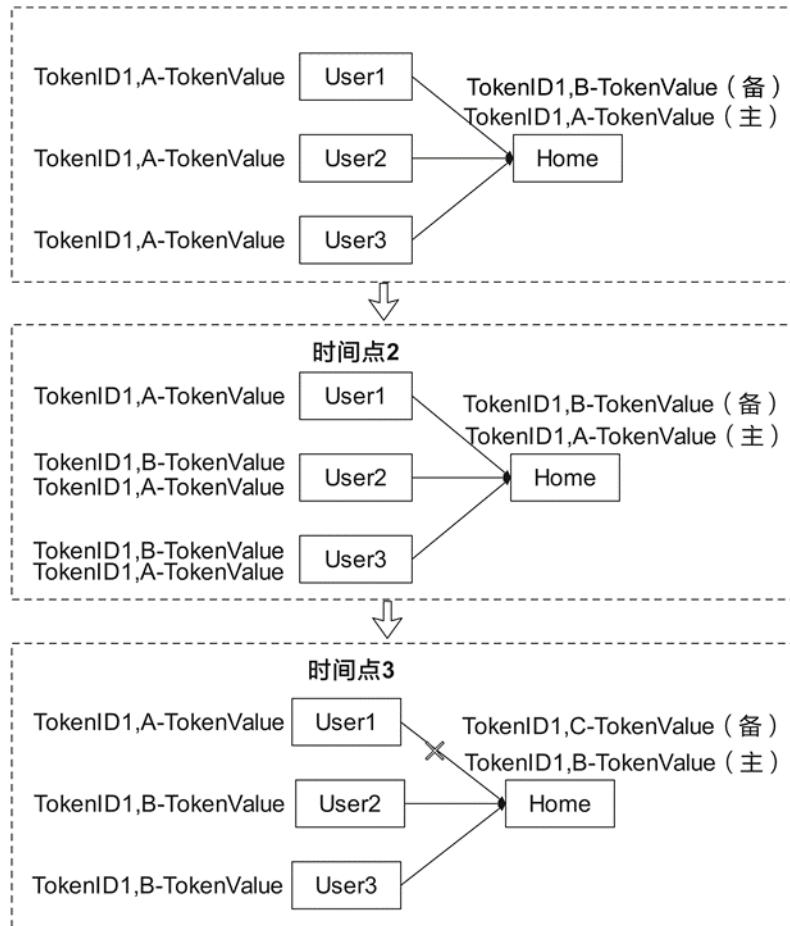


图 11-2 用户粒度权限无效化

## 11.5 数据通路保护

### 11.5.1 应用场景

对于数据通路不可信场景，需对总线上传输的数据包提供基于密码学的机密性和完整性保护（Confidential and Integrity Protection，CIP），抵御窃听、篡改或恶意注入等安全威胁。

注：数据通路不可信，即数据面端到端数据通路（包括物理线路和中间交换转发设备）存在近端物理窃听敏感数据、Switch 主动恶意转发或路由表被恶意篡改使得数据包传到第三方等安全风险。

UB CIP 提供端到端的事务层包粒度的数据通路保护，实现事务层包的机密性、完整性和抗重放攻击保护。

UB CIP 机制仅保护源 Entity 与目的 Entity 之间传输的数据安全,对于该范围以外的数据安全由 UBU 自身安全机制保护。

## 11.5.2 CIP 工作流程

### 11.5.2.1 信道建立流程

CIP 功能核心目的在于保护数据面事务层包,还需要控制面建立 CIP 信道配合。控制面 CIP 信道建立主要是指对源和目的 Entity 的密钥配置;对于集中式方式,可以由域管理者通过带内或带外信道实现密钥配置;对于分布式方式,可以由通信双方的 Entity 进行密钥协商;支持用户自定义其它方式。集中式 CIP 信道建立流程见图 11-3, 步骤描述如下:

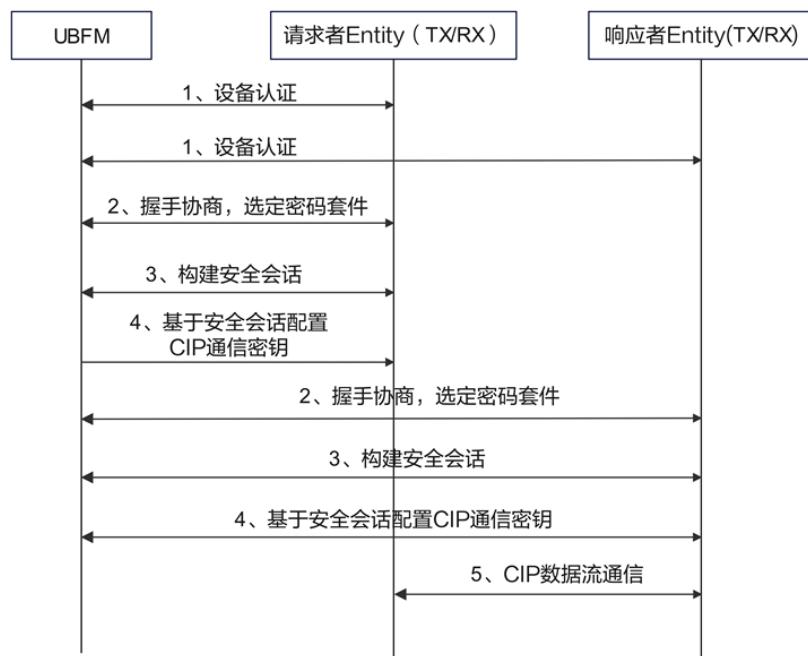


图 11-3 集中式的 CIP 信道建立流程

1. UBFM 分别对请求者 Entity 和响应者 Entity 进行设备认证, 认证过程见 11.2.2 节;
2. UBFM 分别与请求者 Entity 和响应者 Entity 进行握手协商, 确定密码套件;
3. UBFM 分别与请求者 Entity 和响应者 Entity 建立安全会话;
4. UBFM 分别基于安全会话配置请求者 Entity 和响应者 Entity 的 CIP 通信密钥, 包括 TX 和 RX 两个方向;
5. UBFM 分别对 RX ( 可以是请求者或响应者 ) 和 TX ( 可以请求者或响应者 ) 方向的 Entity 完成相应参数配置, 使通信双方的 Entity 可以开始 CIP 数据流通信。

注: 上述步骤 1~4 也可以参考 DSP0274 SPDM 规范。

分布式的 CIP 信道建立流程见图 11-4，步骤描述如下：

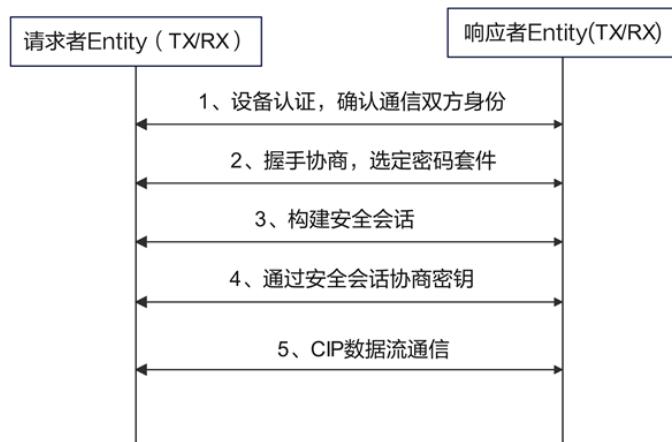


图 11-4 分布式的 CIP 信道建立流程

1. 请求者 Entity 与响应者 Entity 之间完成双向的设备认证，确认通信双方身份，见 11.2.3；
2. 请求者 Entity 与响应者 Entity 握手协商，确定密码套件；
3. 请求者 Entity 与响应者 Entity 建立安全会话；
4. 请求者 Entity 与响应者 Entity 基于安全会话，协商 CIP 通信密钥，密钥协商后，分别通过内部总线，将密钥配置到请求者和响应者 Entity 相关寄存器中，包括发送方 TX 和接收方 RX 两个方向；
5. RX 方向( 可以是请求者或响应者 )和 TX 方向的 Entity 分别通过内部总线做好相应的参数配置，开始 CIP 数据流通信。

注：上述步骤 1~4 也可以参考 DSP0274 SPDM 规范。

### 11.5.2.2 参数取值规则

密码算法选择选用 AES-GCM 或 SM4-GCM 时，算法实现应分别符合 NIST SP 800-38D 或 GM/T 0002-2012。有三个共性关键参数输入，取值建议如下：

- AAD：附加认证数据（Additional Authenticated Data），是指那些需要确保完整性和真实性，但不需要加密的数据。建议选取 CIP 包头、UPI 和事务层包头的部分或全部内容。
- 明文数据：事务层包头中的敏感信息和 PLD 明文。
- IV：初始向量（Initialization Vector），通常是一个 96 bits 的随机数，用于确保每次加密生成不同的密钥流，确保相同的明文使用相同的密钥加密也能产生不同的密文。

### 11.5.2.3 扩展包头

为了支持 CIP 功能，UB 协议栈需在数据包中增加 CIP 安全扩展头，主要包括以下字段：

- CIP ID：本地 Entity 内不同 CIP 策略的索引值。接收方基于 128 bits/20 bits DEID 索引 CIP 策略信息，进一步，利用 CIP ID 索引获得 CIP 包所指示的 CIP ID 所对应的策略信息，包括是否加密、所使用的算法、计算校验所涵盖的字段信息、密钥信息以及 IV 信息等。

注：考虑大规模集群中，CIP 通信流数量很多时候，接收方还可以基于 SEID 或 SEID 和 DEID 建立多层索引，索引 CIP 上下文，包括是否加密、使用的密码算法等，扩展支持更多的 CIP 通信流并存。

- SN：序列号，当支持 CIP 功能时，发送方要为每个数据包生成一个 SN 序列号，填充到 SN 域段中。
- NLP：4 bits，指示下一个包头。
  - (1) 当 CIPH.NLP==4'b0000 时：带 32 bits UPI、128 bits SEID 和 128 bits DEID，带 TAH。
  - (2) 当 CIPH.NLP==4'b0001 时：带 16 bits UPI、20 bits SEID 和 20 bits DEID，带 TAH。
  - (3) 当 CIPH.NLP==4'b0010 时：不带 UPI、SEID 和 DEID，带 TAH。
  - (4) 其它：RSVD。
- RSVD：保留域段。

#### 11.5.2.4 校验方式

当启用 CIP 时，数据包携带完整性校验值（Integrity Check Value，ICV）域段，ICV 的计算依据 CIP 控制寄存器所配置的密码算法，计算数据包中 CIPH 到 PLD 之间域段的消息鉴别码。当前，协议定义当密码算法为 AES-256-GCM 或 SM4-128-GCM 时，ICV 值为 96 bits，建议取消消息鉴别码的低 96 bits。协议支持用户自定义密码算法套件，当密码算法不为 AES-256-GCM 或 SM4-128-GCM 时，ICV 位宽长度依据所选算法确定；通常，ICV 位宽为 32 bits 的整倍数。

CIP 校验仅允许在 ICV 校验成功后才释放事务层包并执行后续操作。在计算该完整性校验 ICV 值的所有事务层包被接收并且完整性检查通过为止，接收方不得释放事务层包，因此接收方需要至少缓存当前事务层包，以确保不会丢失数据。当发生校验错误时，通常建议立即停止该包所请求的操作，以此保证安全强度不降低。

接收方 Entity 校验逻辑如下：

- 校验 CIP 包中的 UPI 与本地 UPI 是否一致，若不一致则拒绝进一步计算。
- 索引该 Entity 的 CIP 相关寄存器，若该 Entity 不启用 CIP，则不进行 CIP 校验，否则基于 CIP ID 索引对应的控制寄存器获取策略信息，包括是否加密、所使用算法、IV 信息、密钥信息、CIP 校验方式、CIP 保护范围等。
- 依据策略信息进行消息鉴别码计算和校验，校验成功后解密 CIP 包中的密文域段并继续执行后续操作。

### 11.5.2.5 错误处理

接收方将对收到的 CIP 包进行预判断处理，将可能产生以下几种情况：

- ICV 校验正确无误：释放事务层包，执行相应操作。
- 在不启用 CIP 机制时接收到 ICV 域段：丢弃事务层包。
- 预期收到 ICV 域段却没收到：丢弃事务层包。
- 其它错误：丢弃事务层包。

### 11.5.2.6 密钥更新

CIP 加密通信密钥的生命周期是有限的，根据实际的加密吞吐情况可以确定密钥更新的周期。本协议支持通过 UBFM 集中配置或通信双方按照协商的密钥更新策略更新 CIP 通信密钥，以确保密钥使用安全。

## 11.6 可信执行环境扩展

### 11.6.1 应用场景

TEE 是基于硬件资源隔离形成的一个安全区域，TEE 中的内存空间和寄存器等，都无法被 TEE 之外的操作系统、虚拟化管理软件等进程访问，结合可信验证和密码技术，确保 TEE 中加载的代码和数据的机密性和完整性都得到保护。

随着大模型训练与推理等高算力需求应用发展，TEE 将逐渐从单处理器扩展至多处理器甚至互联的计算集群。本节将描述如何基于 UB 实现不同 UBPU 中的 TEE 之间安全、高效地数据交互，实现跨 UBPU 的 TEE 扩展，包括基本的概念和模型、核心功能描述和交互流程等。

### 11.6.2 互联模型

#### 11.6.2.1 组件介绍

本节定义 UB Domain 中跨 UBPU 的 TEE 扩展模型，见图 11-5。在该模型中，根据资源访问关系，分为 User 和 Home，User 为访问请求的发起方，Home 为访问请求的响应方。

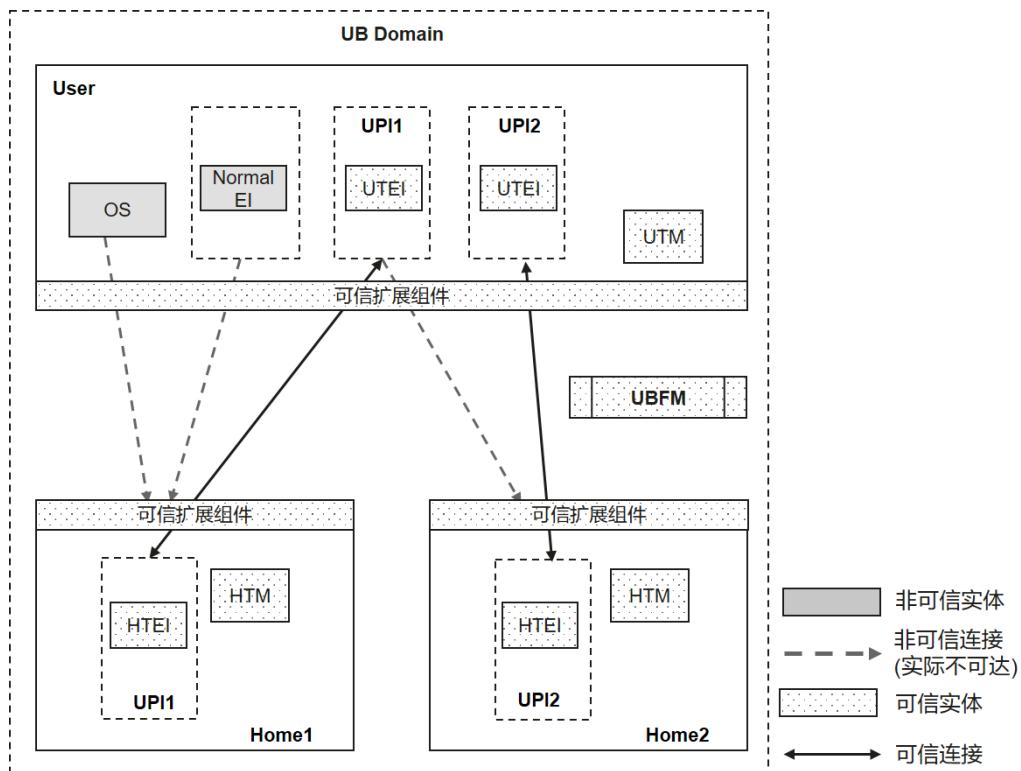


图 11-5 TEE 扩展模型

该互联模型包括以下组件：

- UTEI:** User 侧受 TEE 保护的 Entity 实例，可以是 User 侧运行在 TEE 中使用该 Entity 的可信虚拟机、可信进程或整个 TEE 等。UTEI 作为可信执行环境扩展请求的发起方，通过发送可信扩展数据包和远端 Home 内的 HTEI 建立可信连接，实现跨 UBPU 的资源访问，但不可以访问同一个 Home 内其它未建立可信连接的 HTEI 资源。
- HTEI:** Home 侧受 TEE 保护的 Entity 实例，可以是 Home 侧运行在 TEE 中使用该 Entity 的可信虚拟机、可信进程或者整个 TEE 等。HTEI 作为 TEE 可信扩展请求的响应方，接收和响应 UTEI 发送的可信扩展数据包和远端 User 内的 UTEI 建立可信连接，实现设备间的资源访问，但不可以访问同一个 User 内其它未建立可信连接的 UTEI 资源。
- UTM:** User 侧实现 TEE 可信扩展所必须的 TCB，负责 UTEI 的信任管理和安全策略的正确实施，包括密钥管理、证书管理、可信度量和 UTEI 安全隔离策略等，同时对外向 UBFM 等提供安全服务接口，例如，获取 UTEI 的度量报告，动态授权 (grant) 或撤销 (revoke) 计算资源等。
- HTM:** Home 侧实现 TEE 可信扩展所必须的 TCB，负责 HTEI 的信任管理和安全策略的正确实施，包括密钥管理、证书管理、可信度量和 HTEI 安全隔离策略等，同时对外向 UBFM 等提供可信接口，例如，HTEI 安全状态的管理，获取 HTEI 的度量报告，动态授权或撤销计算资源等。
- 可信扩展组件:** 可信扩展组件是 UB Controller 和 UMMU 中支持 TEE 和 EE\_bits 安全扩展的部分，实现 UTEI 和 HTEI 的安全通信，根据 UBFM 下发的安全配置策略，对 UBPU 向外发送和接收的数据包进行安全管控，即使是 UBPU 内的 UTM 或者 HTM，都无法关闭或者绕过其安全检查。

- **UBFM:** UB Domain 的管理角色，负责设备互联和资源管理，参与 UB Domain 所有 Entity 的安全状态配置
- **OS:** UBPU 中的通用操作系统，为 TEE 之外的 Entity 实例分配系统资源并加载、调度执行。
- **Normal EI:** UBPU 中 TEE 之外的 Entity 实例。

在上述组件中，UTM 或 HTM 配置 Entity 的安全状态，使之能够接入 UTEI 或 HTEI，配置 UMMU，使之能够翻译 TEE UB 地址空间。同一 UPI 内的 UTEI 和 HTEI 之间可以建立安全互联，建议不同业务使用不同的 UPI。

### 11.6.2.2 信任模型

TEE 可信扩展功能依赖以下前提条件：

- UBPU 具备硬件可信根和可信启动能力。
- 存在验证服务器（即本地或远程证明服务器），其中保存有各 UBPU 的固件和操作系统等的完整性校验基线值。

TEE 可信扩展功能存在以下安全假设：

- UPI、UBPU 的硬件可信根和可信启动等都是正确实现且无法被篡改的。
- 验证服务器是可信的，并且其中保存的 UBPU 可信验证基线值是无法被篡改的。
- UTM、HTM、UBFM 和可信扩展组件都是正确设计和实现的，且无法被非授权访问和篡改。

TEE 可信扩展功能目标在于抵御以下安全威胁：

- 对承载 UTEI 或 HTEI 的 UBPU 的身份仿冒、篡改和替换等。
- 对 UTEI 和 HTEI 互连通路上的 UB Switch 的篡改或替换等。
- TEE 之外的组件（例如，UBPU 中的通用 OS、Normal EI 等）对 UTEI 和 HTEI 的非授权访问。
- 不同 UPI 的 UTEI 和/或 HTEI 之间的数据访问。
- 外部攻击者和运维管理人员，对 UTEI 和 HTEI 之间通信数据的窃取、篡改和替换等。

TEE 可信扩展功能不能抵御以下安全威胁：

- 对 UTEI 或 HTEI 的恶意关停、DoS 攻击等破坏可用性的攻击。

### 11.6.2.3 通信模式

TEE 可信扩展功能支持两种模式，分别能够应对的安全威胁见表 11-3。用户可根据设备工作环境和风险接受程度等，灵活选择：

- 基础模式：不考虑物理链路攻击，可不启用 CIP，由 UBFM、UTM 和 HTM 协同给每个 UBPU 上的可信扩展组件配置策略，实施 UB 通信数据流的安全管控，并采用可信验证机制验证 UTEI 和 HTEI 的可信状态。

- 增强模式：考虑更强的威胁假设，即存在物理链路攻击以及对 UBPU 或 UB Switch 的篡改、或替换等攻击，因此，需要在基础模式基础之上，采用 CIP 保护 UTEI 和 HTEI 通信数据的机密性和完整性，并采用可信验证机制验证 UTEI 和 HTEI 的可信状态。

表 11-3 两种互联模式的区别

安全威胁	威胁抵御措施	基础模式	增强模式
对承载 UTEI 或 HTEI 的 UBPU 的篡改和替换	可信验证	√	√
对 UTEI 和 HTEI 互连通路上的 UB Switch 的篡改或替换	可信验证	√	√
TEE 之外的组件（例如，UBPU 中的通用 OS、Normal EI 等）对 UTEI 和 HTEI 的非授权访问	基于 TEE 的安全隔离	√	√
不同 UPI 的 UTEI 和/或 HTEI 之间的数据访问	基于 UB 分区的资源隔离	√	√
外部攻击者和运维管理人员，对 UTEI 和 HTEI 之间通信数据的窃取、篡改和替换等	数据通路保护（CIP）	✗	√

## 11.6.3 通信流程

### 11.6.3.1 基础模式

基础模式下的可信执行环境扩展机制交互流程见图 11-6，具体包括如下四个核心步骤：

#### 1. Entity 实例建立

UTEI 和 HTEI 的资源划分与管理采用资源授权机制，即由 UBFM 发起请求，实现对 TEE 的资源（CPU、内存等）划分，由 UTM 和 HTM 对请求的合法性进行检查，检查通过后，建立 UTEI 或 HTEI。

#### 2. 安全状态配置

UBFM 通过 UTM 和 HTM 对可信扩展组件进行安全配置，包括 UTEI 或 HTEI 安全状态、UTEI 或 HTEI 对应的 UPI 配置和 EE\_bits 配置。

#### 3. UBPU 可信验证

UBPU 可信验证依赖 UBPU 的硬件可信根和可信度量能力，用于验证 UBPU 中的固件、操作系统与 TEE 的完整性。验证方对 UBPU 发起度量挑战，请求中可以引入随机数防止重放攻击。UBPU 响应度量挑战请求，对 UBPU 中需要验证的信息进行度量并基于私钥对度量值签名。验证方收到签名的度量值后，与验证服务器中存储的基线值进行比对。若一致，说明 UBPU 中的固件与 TEE 环境符合预期，可执行后续任务。

User 和 Home 度量验证的对象至少包括：UBPU 固件（包含设备驱动和 UTM 或 HTM）、OS 内核、文件系统和 UTEI 或 HTEI 的安全状态配置信息等。

如果 UB Switch 位于 User 与 Home 之间的互联链路上，也需要对 UB Switch 进行度量验证，度量验证的对象包括 UB Switch 固件等。

验证方通过 UTM 或 HTM 对 UTEI 或 HTEI 进行可信验证，验证通过后，UTEI 与 HTEI 才可通信。通信过程中，UTEI 或 HTEI 发送的数据包将携带 EE\_bits。

#### 4. TEE 可信扩展

通信过程中，由可信扩展组件根据安全策略，对数据包进行访问控制过滤，依赖 UMMU 和 UB Decoder 对 EE\_bits 的硬件扩展。

基于 EE\_bits 进行访问控制判断时，应遵循 UMMU 对内存的访问管理过程，见内存管理 9.4 节。

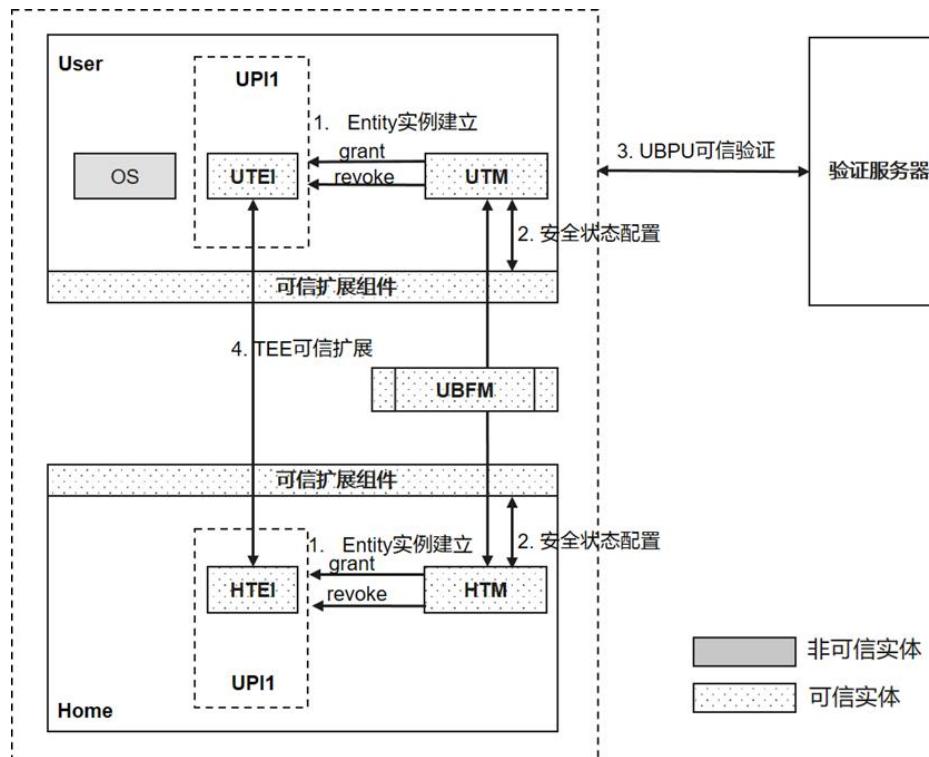


图 11-6 TEE 扩展过程

#### 11.6.3.2 增强模式

增强模式下，需要考虑物理攻击的安全威胁，可信执行环境之间安全通信具体步骤包括：

##### 1. Entity 实例建立

与基础模式一致，见 11.6.3.1 节 1。

##### 2. 安全状态切换

与基础模式一致，见 11.6.3.1 节 2。

##### 3. 设备可信验证

与基础模式一致，见 11.6.3.1 节 3。

#### 4. TEE 可信扩展

在基础模式的基础之上（见 11.6.3.1 节 4），见 11.5.2.1 节，由 UBFM 对 UBPU 进行密钥分发和配置，启用 CIP 功能，保护 UTEI 和 HTEI 通信数据的机密性和完整性。

### 11.6.4 内存地址隔离

可信扩展组件在数据包中增加 EE\_bits 域段（域段定义见 7.2.1 节），并基于 EE\_bits 对 UB 地址空间进行安全隔离，根据实际系统设计的不同，EE\_bits 最大允许 2 比特，即可划分 4 个地址空间，其中一个属于非 TEE UB 地址空间，一个属于 TEE 地址空间，其它两个地址空间预留，可供用户扩展使用。

为了实现不同安全状态 Entity 的内存地址空间隔离，需要将 UB 地址空间划分成两部分：非 TEE UB 地址空间和 TEE UB 地址空间，如图 11-7。

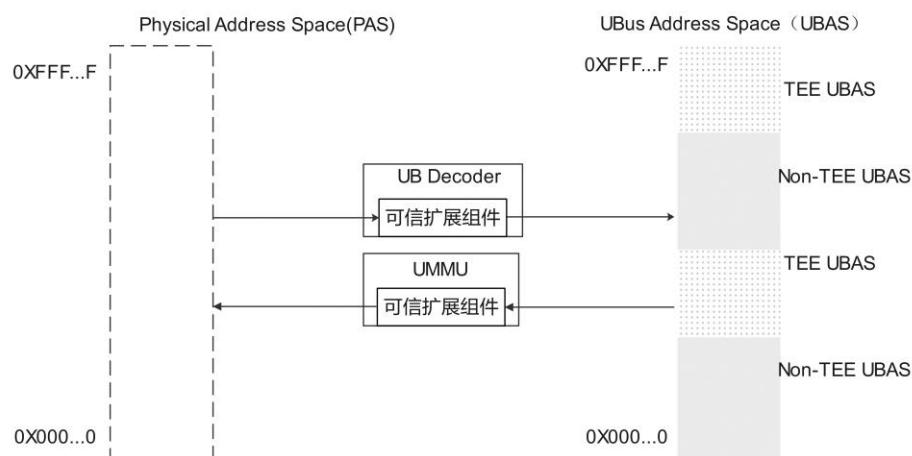


图 11-7 UB 地址空间隔离

需要注意的是，非 TEE 和 TEE UB 地址空间是带上不同 EE\_bits 的 UB 地址空间，任意一个 UB 地址，任意时刻只能属于非 TEE UB 地址空间或者 TEE UB 地址空间，不能同时属于两者。

实现 UB 地址空间安全隔离主要包括以下两方面的协议扩展：

- User 发送数据包时，在 User 发起内存访问的数据包中增加 EE\_bits，用于标识 User 侧发送数据包的进程所在执行环境的安全状态。
- Home 接收数据包时，根据数据包中的 EE\_bits 选择 UMMU 页表，UMMU 支持对 TEE UB 地址空间的寻址。

### 11.6.5 配置流程参考

图 11-8 给出了云计算场景下 TEE 扩展的一种参考配置流程，用户通过 UBFM 配置 TEE 扩展流程主要分为五步：

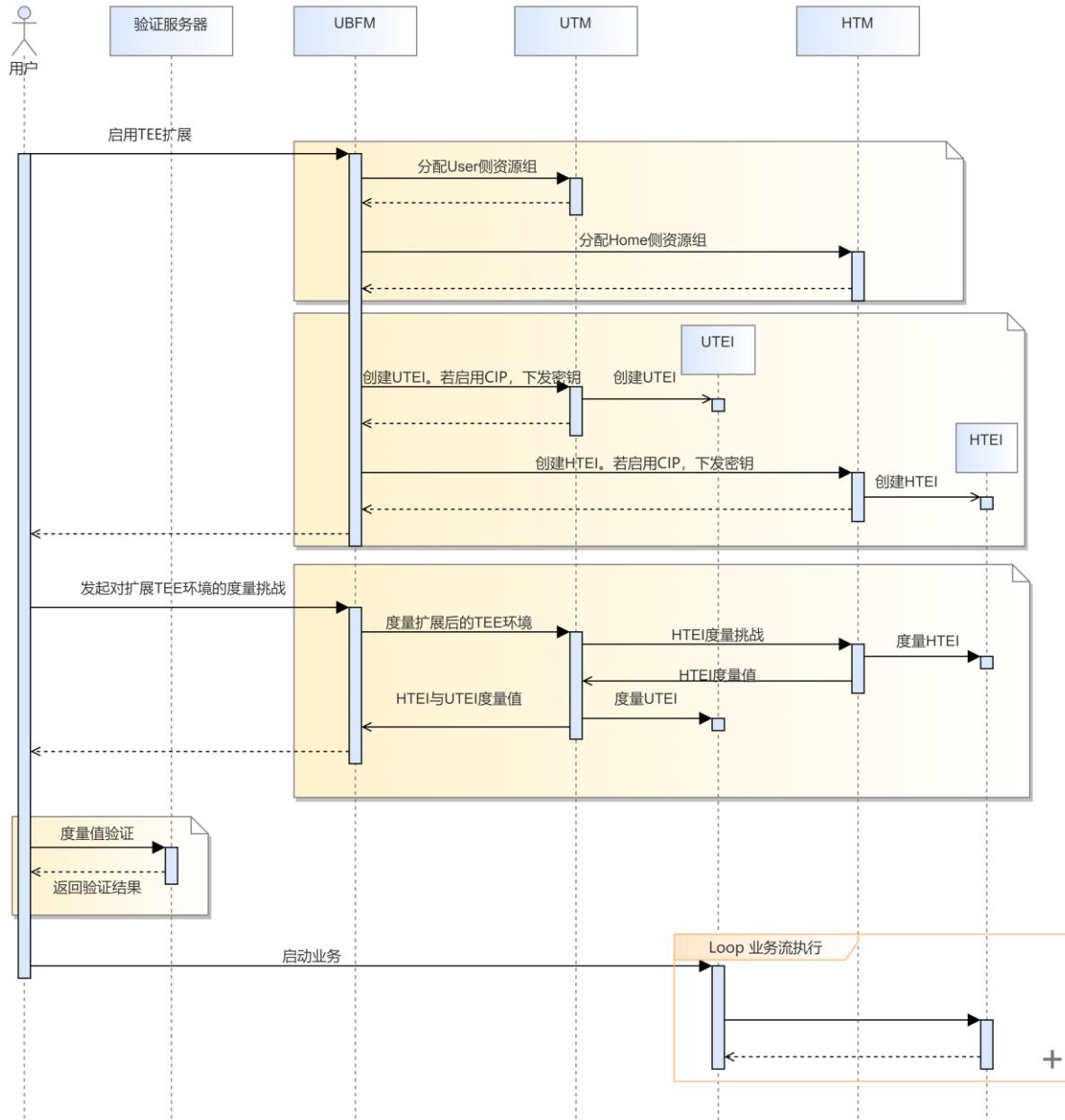


图 11-8 TEE 扩展配置流程参考

1. 资源配置：创建 User 侧或 Home 侧所需资源，配置 User 与 Home 之间的互联信道。
2. 创建 UTEI、HTEI：配置 UTEI、HTEI 安全状态，配置 User 侧与 Home 侧 UMMU、UB Decoder 寄存器页表等。若用户启用增强模式，见 11.5.2.1 节为 UTEI 和 HTEI 配置通信密钥。
3. 用户对 TEE 环境发起度量挑战。
4. 用户获取度量值之后，发送给验证服务器验证度量值。

5. 用户完成度量值验证后，开始正式业务流。

本配置流程只作为参考流程，在实际业务中可以根据业务需求在保证方案安全性与完备性的前提下，进行方案实现。如：

3.项，用户可通过 UBFM 分别获取 UTEI 和 HTEI 的度量值。适用于 UBFM 分别与 User 和 Home 侧建立安全信道的场景。

4.项，UTEI 和 HTEI 可直接将度量值返回给验证服务器，验证服务器验证完成后，向用户返回验证结果。

# 附录 A 缩略语

缩略语	全称
Addr	地址 ( Address )
AE	异步事件 ( Asynchronous Event )
AMCTL	帧定界控制码字 ( Alignment Marker Control )
AR	自适应路由 ( Adaptive Routing )
BCRC	块循环冗余校验 ( Block Cyclic Redundancy Check )
BER	误码率 ( Bit Error Ratio )
BMC	主板管理控制单元 ( Baseboard Management Controller )
BMS	裸金属服务器 ( Bare Metal Server )
CC	拥塞控制 ( Congestion Control )
CFG	配置 ( Configuration )
CI	消费者索引 ( Consumer Index )
CIP	机密性和完整性保护 ( Confidentiality and Integrity Protection )
CNA	简短网络地址 ( Compact Network Address )
CO	完成序 ( Completion Order )
CPU	中央处理单元 ( Central Processing Unit )
CQ	完成队列 ( Completion Queue )
CQE	完成队列元素 ( Complete Queue Element )
CRC	循环冗余校验 ( Cyclic Redundancy Check )
DCNA	目的简短网络地址 ( Destination Compact Network Address )
DEID	目的 UB 实体身份标识 ( Destination UB Entity Identifier )
DGUID	目的全局唯一标识符 ( Destination Globally Unique Identifier )
DHCP	动态主机配置协议 ( Dynamic Host Configuration Protocol)
DLLCB	数据链路层控制块 ( Data Link Layer Control Block )
DLLDB	数据链路层数据块 ( Data Link Layer Data Block )

缩略语	全称
DLLDP	数据链路层数据包 ( Data Link Layer Data Packet )
DMA	直接内存访问 ( Direct Memory Access )
DPU	数据处理单元 ( Data Processing Unit )
DSCP	区分服务编码点 (Differentiated Services Code Point)
DW	双字 ( Double Word )
ECN	显式拥塞通知 (Explicit Congestion Notification)
EDF	结束发送微片的标志 ( End of Data Flit )
EE_bits	执行环境标识位 ( execution environment bits )
EID	实体身份标识 ( Entity identifier )
EN	启用 ( Enable )
EO	仅存在 ( Exist Only )
EQ	事件队列 ( Event Queue )
EQE	事件队列元素 ( Event Queue Element )
ESN	设备序列号 ( Equipment Sequence Number )
Exec	执行 ( Execute )
EXT	扩展 ( Extend )
FEC	前向纠错 ( Forward Error Correction )
FECN	前向显式拥塞通知 (Forward Explicit Congestion Notification)
GB	吉字节 ( Gigabyte )
GPU	图形处理单元 ( Graphics Processing Unit )
GUID	全局唯一身份标识 ( Globally Unique Identifier )
Hdr	协议头 ( Header )
HSM	硬件安全模块 ( Hardware Security Module )
HTEI	Home 侧可信实体的实例 ( Trusted Entity Instance in Home )
HTM	Home 侧的可信执行环境管理组件 ( Trusted execution environment Manager in Home )
HwInit	硬件初始化 ( Hardware Initialization )
IANA	因特网编号分配机构 (Internet Assigned Numbers Authority)

缩略语	全称
ICMP	互联网控制报文协议 ( Internet Control Message Protocol )
ICV	完整性校验值 ( Integrity Check Value )
ICRC	不变循环冗余校验 ( Invariant Cyclic Redundancy Check )
ID	身份标识 ( Identification )
IETF	互联网工程任务组 (Internet Engineering Task Force)
IF	界面 ( Interface )
IHL	首部长度 (Internet Header Length)
IP	互联网协议 ( Internet Protocol )
IPA	中间物理地址 ( Intermediate Physical Address )
IPAS	中间物理地址位宽 ( Intermediate Physical Address Size )
IPSEC	互联网安全协议 (Internet Protocol Security)
IV	初始向量 ( Initialization Vector )
JFC	用于存放 Jetty 事务的完成事件 ( Jetty For Completion )
JFCE	用于触发完成事件中断 ( Jetty For Completion Event )
JFR	用于接收的 Jetty ( Jetty For Receiving )
JFS	用于发送的 Jetty ( Jetty For Sending )
KB	千字节 ( KiloByte )
LBC	灵衢块设备命令集 ( Linquick Block Command Set Specification )
LBF	负载均衡因子 ( Load Balance Factor )
LBH	数据链路层块头 ( Link Block Header )
LoC	位置 ( Location )
LPC	灵衢持久化日志存储设备命令集 ( Linquick PLOG Command Set Specification )
LPH	数据链路层包头 ( Link Packet Header )
MAPT	内存访问权限表 ( Memory Address Permission Table )
MAPTE	内存访问权限表表项 ( Memory Address Permission Table Entry )
MATT	内存访问转换表 ( Memory Address Translation Table )
MATTE	内存访问转换表表项 ( Memory Address Translation Table Entry )

缩略语	全称
MMIO	内存映射输入输出 ( Memory-mapped I/O )
MSN	消息序号 (Message Sequence Number)
MTU	最大传输单元 ( Maximum Transfer Unit )
NLP	下层协议指示 ( Next Layer Protocol )
NO	无序 ( No Order )
NoC	片上网络 ( Network on Chip )
NOS	网络操作系统 (network operating system)
NPI	网络分区识别符 ( Network Partition Identifier )
NPIC	网络分区识别符配置 ( Network Partition Identifier Configuration )
NPU	神经网络处理单元 ( Neural Processing Unit )
NTH	网络层包头 ( Network Header )
PA	物理地址 ( Physical Address )
PAS	物理地址位宽 ( Physical Address Size )
PCS	物理编码子层 ( Physical Coding Sublayer )
PDU	协议数据单元 ( Protocol Data Unit )
PI	生产者索引 ( Producer Index )
PLB	权限旁路缓存 ( Permission Lookaside Buffer )
PMA	物理媒介适配子层 ( Physical Medium Attachment )
PSN	包序列号 ( Packet Sequence Number )
PTE	页表表项 ( Page Table Entry )
QoS	服务质量 ( Quality of Service )
RC	请求者上下文 ( Requester Context )
RCID	请求者上下文标识符 ( Requester Context Identifier )
REQ	请求 ( Request )
RNR	接收端未就绪(Receiver Not Ready)
RO	弱序 ( Relaxed Order )
ROE	资源拥有者实体 ( Resource-Owner Entity )

## 附录 A 缩略语

缩略语	全称
ROI	可靠和发送端保序模式 ( Reliable and Ordered by Initiator )
ROL	可靠和下层保序模式 ( Reliable and Ordered by Lower Layer )
ROS	粘性只读 ( Read Only Sticky )
ROT	可靠和目的端保序模式 ( Reliable and Ordered by Target )
RQ	接收队列 ( Receive Queue )
RQE	接收队列元素 ( Receive Queue Element )
RSP	响应 ( Response )
RST	复位 ( Reset )
RSVD	保留 ( Reserved )
RT	路由类型 ( Routing Type )
RUE	资源使用者实体 ( Resource-User Entity )
RW	读写 ( Read Write )
RW1C	读写且写 1 清除 (Read Write 1 to Clear)
RW1CS	粘性读写且写 1 清除 (Read Write 1 to Clear Sticky)
RWS	粘性读写 ( Read Write Sticky )
RX	接收器 ( Receiver )
SC	序列上下文 ( Sequence Context )
SCID	序列上下文标识符 ( Sequence Context ID )
SCNA	源简短网络地址 ( Source Compact Network Address )
SDF	开始发送 Flit 的标志 ( Start of Data Flit )
SerDes	串行器/解串器 ( Serializer/Deserializer )
SGE	散集元素 ( Scatter Gather Element )
SGL	散集列表 ( Scatter Gather List )
SGUID	源全局唯一标识符 ( Source Globally Unique Identifier )
SL	服务优先级 ( Service Level )
SLFT	分段式线性路由表 ( Segment Linear Forward Table )
SO	强序 ( Strong Order )

缩略语	全称
SoC	片上系统 (System on Chip)
SQ	发送队列 (Send Queue)
SQE	发送队列元素 (Send Queue Element)
SSU	可扩展存储单元 (Scalable Storage Unit)
TAACK	事务响应 (Transaction Acknowledgement)
TAH	事务层包头 (Transaction Header)
TASSN	事务编号 (Transaction Segment Sequence Number)
TC	流类型 (Traffic class)
TCB	可信计算基 (Trusted Computing Base)
TCT	目标上下文表 (Target Context Table)
TCTE	目标上下文表项 (Target Context Table Entry)
TECT	目标实例配置表 (Target Entity Configuration Table)
TECTE	目标实例配置表表项 (Target Entity Configuration Table Entry)
TEE	可信执行环境 (Trusted Execution Environment)
TLB	转换旁路缓存 (Translation Lookaside Buffer)
TLV	类型长度值 (Type-Length-Value)
TOS	服务类型 (Type of Service)
TP	传输 (Transport)
TP Packet	传输层数据包 (Transport data Packet)
TPACK	传输正确应答 (Transport Acknowledgement)
TPEP	传输端点 (Transport Endpoint)
TPG	传输通道组 (Transport Channel Group)
TPH	传输层包头 (Transport Header)
TPNAK	传输错误应答 (Transport Negative Acknowledgement)
TPSACK	传输选择性应答 (Transport Selective Acknowledgement)
LTB	训练符号块 (Link Training Block)
TTL	生存时间 (Time to Live)

## 附录 A 缩略语

缩略语	全称
TX	发送器 ( Transmitter )
UB2E	UB 转以太网 ( UB to Ethernet )
UBA	UB 地址 ( UB Address )
UBAS	UB 地址空间 ( UB Address Space )
UBFM	灵衢互连结构管理器 ( UB Fabric Manager )
UBMD	UB 内存描述符 ( UB Memory Descriptor )
UBoE	UB 承载以太网 ( UB over Ethernet )
UDP	用户数据报协议 ( User Datagram Protocol )
UEID	用户的实体身份标识 ( User Entity identifier )
ULAP	UB 链路聚合协议 ( UB Link Aggregation Protocol )
UMAPTE	唯一内存访问权限表表项 ( Unique Memory Address Permission Table Entry )
UMMU	UB 内存管理单元 ( UB Memory Management Unit )
UNO	不可靠无保序模式 ( Unreliable and Non-Ordering )
UPI	UB 分区标识符 ( UB Partition Identifier )
URMA	统一远程内存访问 ( Unified Remote Memory Access )
USI	UB 信号中断 ( UB Signaled Interrupt )
UTEI	User 内可信实体的实例 ( Trusted Entity Instance in User )
UTM	User 侧的可信执行环境管理组件 ( Trusted execution environment Manager in User )
VL	虚拟通道 ( Virtual Lane )
VM	虚机 ( Virtual Machine )
VMM	虚拟机监视器 ( Virtual Machine Monitor )
VMS	虚机服务 ( Virtual Machine Service )

# 附录 B 包格式

## B.1 概述

本章列出 UB 支持的各种包格式排布（若某种包的格式排布包含在相关章节，则本节不列出对应包格式，而是指向相关章节），不包括各层包头格式及域段定义。各层包头格式，域段定义及使用说明请参考相关协议层章节。

UB 包格式根据数据链路层包头中的 CFG 域段值分成几种类型，有些类型根据协议包头域段进一步细分，如下表：

表 B-1 基于 LPH.CFG 值的包分类

CFG	包类型判别	说明
0	CFG==0	数据链路层控制块，用于 UB 链路两个端口之间的控制，状态及配置信息交互。
3	( CFG==3 ) & ( ( 非 UDP 包 )   ( UDP 目的 Port != 4792 ) )	TCP/IP 协议栈包，基于 IPv4。
	( CFG==3 ) & ( UDP 目的 Port == 4792 )	UB 包，基于 IPv4。
4	( CFG==4 ) & ( ( 非 UDP 包 )   ( UDP 目的 Port != 4792 ) )	TCP/IP 协议栈包，基于 IPv6。
	( CFG==4 ) & ( UDP 目的 Port == 4792 )	UB 包，基于 IPv6。
5	CFG==5	网络控制包。
6	( CFG==6 ) & ( ( NTH.NLP==0 ) & ( BTAHOpcode!=0x10 ) )	用于 Non-Coherent 内存空间及设备资源空间访问。
	( CFG==6 & ( ( NTH.NLP==0 ) & ( BTAHOpcode==0x10 ) ) )	配置管理包，用于配置空间访问，资源管理等各类消息。
	( CFG==6 ) & ( NTH.NLP==1 )	枚举包。
7	CFG==7	UB 包，基于网络层 CNA 地址。
9	CFG==9	用于 Coherent 内存空间访问。
1,2,8, 10~15	Reserved	Reserved。

## B.2 包格式

各部分包头的颜色如下，除特殊说明外，适用于本节所有包格式：

LPH 数据链路层包头 ( LPH )：定义详见 4.3.2.2 数据链路层 LPH 格式。

NTH 网络层包头 ( NTH )：根据组网规模或通信对象可选 IP 包头，24-bit CNA 包头，16-bit CNA 包头，详见 5.2 网络层包头。

TPH 传输层包头 ( TPH )：根据传输层模式可选 RTPH，UTPH，CTPH，详见 6.2.1 节。

UPIH EIDH CIPH UPIH ( 分为 32 bits 和 16 bits )，EIDH ( 分为 256 bits 和 40 bits )，定义详见 B.3 和 B.4 节，CIPH 详见 11.5.2.3 安全扩展包头。

TAH 事务层包头 ( TAH )：包括基础事务层包头 BTAH 以及各种扩展事务层包头，在各种场景以及语义下包括的扩展事务包头不一样，详见 7.2 事务层包头。

注 1：在各种包格式中，对于 Pad，Reserved ( 或者 Rsvd ) 域段，在发送端用全 0 填充；接收侧忽略该域段值。

### B.2.1 LPH.CFG==0 包格式

LPH.CFG==0 的包是数据链路层控制块。数据链路层控制块用于链路两端端口的控制，状态及配置信息交互，实现数据链路层流控，重传，端口信息交互及其它功能。数据链路层控制块由数据链路层产生及处理，不包括 NTH、TPH、TAH。包格式及定义详见 4.3.3 数据链路层控制块 ( DLLCB ) 格式部分。

### B.2.2 LPH.CFG==3/4 包格式

#### B.2.2.1 基于 IP 地址的 UB 包格式

基于 IP 地址的 UB 包的 LPH.CFG 值等于 3 ( IPV4 ) 或 4 ( IPV6 )，其 NTH 采用 IP 包头，修改和新定义了部分网络互连规则，详见网络层；格式中包括 UDP 包头，UDP 的目的端口号等于知名端口号 4792；其传输层使用 RTP 或者 UTP 模式，包格式中的 TPH 为 RTPH 或者 UTPH。

依据物理机或者虚拟机通信，其包格式可选包括隔离 ( UPIH ) 及身份识别 ( EIDH ) 域段。若包含 UPIH，则 UPI 长度为 32 位；若包含 EIDH，则包括 128 位源 EID 和 128 位目的 EID。UPIH 和 EIDH 域段参见本章的 UPIH ( 见 B.3 节 ) 和 EIDH ( 见 B.4 节 ) 格式。

基本格式如下，其中 TPH.NLP==0：

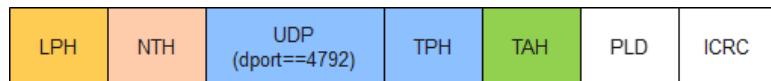


图 B-1 基于 IP 的 URMA 基本包格式

包括 UPIH 和 EIDH 域段，用于虚拟化场景，其中 TPH.NLP==1：



图 B-2 基于 IP 的 URMA 带 UPIH 和 EIDH 包格式

TPH 之后跟着安全头 CIPH，其中 TPH.NLP==3，CIPH.NLP==2：



图 B-3 基于 IP 的 URMA 带 CIPH 包格式

CIPH 后带 UPIH 和 EIDH，其中：TPH.NLP==3，CIPH.NLP==0：



图 B-4 基于 IP 的 URMA 带 CIPH, UPIH, EIDH 包格式

### B.2.2.2 TCP/IP 包格式

UB 支持 TCP/IP 协议栈，NTH 采用 TCP/IP 协议栈网络层包头，LPH.CFG==3 时使用 IPv4 包头；LPH.CFG==4 时使用 IPv6 包头。其包格式详见网络层 5.2.3。

### B.2.3 LPH.CFG==5 包格式

LPH.CFG==5 的包是网络控制包。

网络控制包支持 UB 网络对接 IP 网络管理协议，其格式详见附录 F.2。

### B.2.4 LPH.CFG==6 包格式

#### B.2.4.1 Non-Coherent 内存空间访问包格式

LPH.CFG==6 的包用于 Non-Coherent 内存空间及设备资源空间访问，其 NTH 采用基于 16-bit CNA 的网络层包头，NTH.NLP==0。

此类型包的传输层采用 CTPH 模式，CTPH.NLP==2。

本小节包使用的 UPI 长度为 16 位；EIDH 包括 20 位源 EID 和 20 位目的 EID。UPI 和 EID 域段参见本章的 UPIH 和 EIDH 格式。



图 B-5 Non-Coherent 内存空间访问包格式

带 CIPH, UPIH, EIDH ( CTPH.NLP==3, CIPH.NLP==1 ) :



图 B-6 Non-Coherent 内存空间访问带 CIPH 包格式

### B.2.4.2 配置管理包格式

枚举包用于 UBFM 进行枚举及简短网络地址 ( CNA ) 分配及查询。

枚举包的 LPH.CFG==6, 其 NTH.NLP==1。

配置管理包格式参考 10.4.3 资源管理章节中的管理命令部分。

### B.2.4.3 枚举包格式

枚举包用于 UBFM 进行枚举及简短网络地址 ( CNA ) 分配及查询。

枚举包的 LPH.CFG==6, 其 NTH.NLP==1。

枚举包格式参考 10.4.3 资源管理章节中的管理命令部分。

## B.2.5 LPH.CFG==7 包格式

LPH.CFG==7 的包是基于 24-bit CNA 网络层地址的 UB 包, 其 NTH 采用基于 24-bit CNA 的网络层包头。

此类包若包含 UPIH, 则 UPI 长度为 32 位; 若包含 EIDH, 则包括 128 位源 EID 和 128 位目的 EID。 UPIH 和 EIDH 域段参见本章的 UPIH 和 EIDH 格式。

根据 NTH.NLP 值的不同可以带 CTPH 或 RTPH, 参见 5.2.2 节。

**基于 24-bit CNA 地址及 CTP 的 UB 包格式 ( NTH.NLP==3'b000 ):**

CTPH.NLP==1, 带 UPIH 和 EIDH:



图 B-7 基于 24-bit CNA 地址+CTPH+UPIH+EIDH 包格式

CTPH.NLP==3, 带 CIPH, 通过 CIPH 指示后续包格式:



图 B-8 基于 24-bit CNA 地址+CIPH+CTPH+UPIH+EIDH 包格式

#### 基于 24-bit CNA 地址及 RTP 的 UB 包格式 ( NTH.NLP==3'b010 ):

除使用基于 24-bit CNA 地址的 NTH 外，此模式的包格式种类及各域段含义都和基于 IP 地址的 UB 包格式 ( LPH.CFG==3,4 ) 相同，参考基于 IP 地址的 UB 包格式 ( LPH.CFG==3,4 )。

### B.2.6 LPH.CFG==9 包格式

LPH.CFG==9 的包使用 16-bit CNA 网络头 ( NTH ), NTH.NLP==0, 无 ICRC。

此类包无传输层包头，无 EIDH，无 UPIH。

此小节按照请求和响应列出 LPH.CFG==9 的包格式，不同事务操作下 TAH 包括的具体扩展头类型及格式详见事务层。

Request with payload 包格式 ( Write, Write with BE, Atomic ):

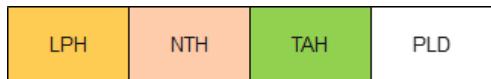


图 B-9 CFG9 Request with payload 包格式

Request without payload 包格式 ( Read, PrefetchTgt ):

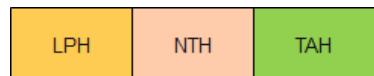


图 B-10 CFG9 Request without payload 包格式

Response with payload 包格式 ( Read response, Atomic Response ( 除 Atomic Store Response 之外 ) ):

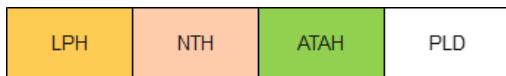


图 B-11 CFG9 Response with payload 包格式

Response without payload 包格式 ( Write Response, Atomic Store Response ):

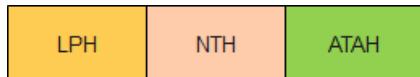


图 B-12 CFG9 Response without payload 包格式

### B.2.7 传输层包格式

传输层包用于传输层本身对业务包的响应及拥塞通知，由传输层产生及处理。

CTP 模式下仅实现拥塞控制功能，只支持 CNP 包；对于请求不需要返回 TPACK，因此不支持各种 TPACK 包。

RTP 模式下支持所有类型的传输层包。

其格式参考传输层 6.2 章节。

### B.3 UPI Header (UPIH)

UPI 是若干个 Entity 集合的隔离标识符，承担 Entity 分区访问隔离功能，在 Domain 内具有唯一性，不同 UPI 标示的 Entity 分区不能互相通信。UPI 不参与交换机转发。接收侧 UPI 的校验规则如下：接收包时，本地的 UPI 与接收的包中携带的 UPI 进行比较，两者相同才允许访问本地资源。

UPIH 格式和定义如下：

Byte0								Byte1								Byte2								Byte3							
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
UPI[31:0]																															

图 B-13 UPIH 32 Bits 格式

Byte0								Byte1							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
UPI[15:0]															

图 B-14 UPIH 16 Bits 格式

表 B-2 UPIH 域段

域段名	位宽(bit)	描述
UPI	32/16	<p>UPI 有两种格式，定义如下：</p> <ul style="list-style-type: none"> <li>非压缩格式：           <ul style="list-style-type: none"> <li>UPI[23:0]: 代表分区号；</li> <li>UPI[31:24]: Reserved。</li> </ul> </li> <li>压缩格式：           <ul style="list-style-type: none"> <li>UPI[14:0]: 代表分区号；</li> <li>UPI[15]: Reserved。</li> </ul> </li> </ul>

## B.4 EID Header ( EIDH )

EID 头格式和定义如下：

Byte0								Byte1								Byte2								Byte3							
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
SEID[127:96]																															
SEID[95:64]																															
SEID[63:32]																															
SEID[31:0]																															
DEID[127:96]																															
DEID[95:64]																															
DEID[63:32]																															
DEID[31:0]																															

图 B-15 EIDH 128 Bits 格式

Byte0								Byte1								Byte2								Byte3							
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
SEID[19:0]																DEID[19:8]															
DEID[7:0]																															

图 B-16 EIDH 20 Bits 格式

表 B-3 EIDH 域段

域段名	位宽(bit)	描述
SEID	128/20	Source EID，有两种格式： • 非压缩格式：128 Bits • 压缩格式：20 Bits
DEID	128/20	Destination EID，有两种格式： • 非压缩格式：128 Bits • 压缩格式：20 Bits

# 附录 C GUID 和 Class Code 编码

## C.1 GUID 编码定义

Globally Unique Identifier ( GUID ) 是 Entity 的永久身份标识，每个 Entity 都具有一个 GUID，在生产制造时生成，具备全局唯一性。GUID 长度为 128 bits，值为 0 时表示无效，结构如下图所示：

域段	Vendor ID	Device ID	Version	Type	Reserved	Sequence Number
位宽 (bit)	16	16	4	4	24	64

图 C-1 GUID 结构

1. Sequence Number: 64 bits, 序列号, 由厂商自定义。
2. Reserved: 24 bits, 协议保留。
3. Device ID: 16 bits, 设备标识, 由厂商自定义。
4. Version: 4 bits, 设备版本信息的编码, 由厂商自定义。
5. Type: 4 bits, 设备类型的编码, 由 UB 协议定义。Type 域段编码定义:
  - (1) 0x0: Bus Instance
  - (2) 0x1: independent UB Controller
  - (3) 0x2: integrated UB Controller
  - (4) 0x3: independent UB Switch
  - (5) 0x4: integrated UB Switch
  - (6) 0x5~0xF: 保留
6. Vendor ID: 16 bits, 厂商唯一标识, 由协议归属组织统一分配。

UB 设备可实现为不同的物理形态，可以是独立的 SoC ( independent type )，也可以被集成在 SoC 内 ( integrated type )，也可以是虚拟化管理软件栈等软件生产的设备 ( 如 Bus Instance )。UB 设备的物理类型在 Type 编码中描述，一个 UB 设备内的多个 Entity 应是相同的物理类型。

注：本规范保留部分 GUID 值作为知名 GUID 使用，知名 GUID 的高 112 bits 为全 1。Entity 不能占用知名 GUID。

## C.2 Class Code 编码定义

Class Code 标识 Entity 声明的功能或服务，分为 Base Code 和 Sub Code，各 8 bits。

Class Code [15:0]		
Base Code [7:0]	Sub Code [15:8]	描述
0x00 总线控制器	0x00	UB 总线控制器
	0x01-0xFF	保留
0x01 存储控制器	0x00	LPC 存储控制器
	0x01	LBC 存储控制器
	0x02-0xFF	保留
0x02 网络控制器	0x00	UB 网络控制器
	0x01	以太网控制器
	0x02-0xFF	保留
0x03 交换设备	0x00	UB 交换设备
	0x01-0xFF	保留
0x04 神经网络设备	0x00	UB 神经网络设备
	0x01-0xFF	保留
0x05-0xFE	0x00-0xFF	保留
0xFF 其它功能的设备	0x00	未知功能的设备
	0x01-0xFF	保留

## C.3 编码使用

### C.3.1 UB Switch

UB Switch 作为一种实现 UB 协议栈的组件，可以被集成在 UBPU SoC 内，也可以实现为独立的 SoC，使用 Class Code 声明其提供的数据包路由服务。系统软件可根据 GUID 的 Type 编码为 0x3 或 0x4 识别 UB Switch 组件，可根据 Class Code 识别其具体的数据包路由服务。

### C.3.2 UB Controller

UB Controller 作为一类实现 UB 协议栈的组件，被集成在 UBPU SoC 内，其 Entity 使用 Class Code 声明 SoC 可提供的功能或服务，系统软件可根据 GUID 的 Type 编码和 Class Code 编码进行识别。

总线控制器是 UB Controller 的一种具体实现，集成在 UBPU SoC 内，可为 SoC 提供同步访问远端内存和收发管理消息的能力，是 UB 设备进入 OS 的初始入口，系统软件可根据 GUID 的 Type 编码为 0x2 且 Class Code 编码为 0x0000 识别 UB 总线控制器。

虚拟化管理软件栈可基于指定的一个或多个总线控制器创建总线控制器虚拟实例，并将其分配给虚机使用。总线控制器虚拟实例是 UB 设备进入虚机 OS 的初始入口，使虚机可与物理世界中的设备互通。总线控制器虚拟实例具有一个虚拟化 Entity ( virtualized Entity )，支持被分配 EID 和 UPI，共享对应总线控制器的端口。总线控制器虚拟实例在被创建时应拥有一个 GUID，由管理员根据 GUID 定义规则为其生成一个 GUID，系统软件可根据 GUID 的 Type 编码为 0x0 识别总线控制器虚拟实例。

# 附录 D 配置空间寄存器

## D.1 CFG0\_BASIC

### D.1.1 整体结构

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
																															0X0	
																															0X1	
																															0X2	
																															0X3	
																															0X4	
																															0X5	
																															0X6	
																															0X7	
																															0X8	
																															0X9	
																															0XA	
																															0XB	
																															0XC	
																															0XD	
																															0XE	
																															0XF	
																															0X10	
																															0X11	
																															0X12	
																															0X13	
																															0X14	
																															0X15	
																															0X16	
																															0X17	
																															0X18	
																															0X19	
																															0X1A	
																															0X1B	
																															0X1C	
																															0X1D	
																															0X1E	
																															0X1F	
																															0X20	
																															DEV_RST	
																															0X21	
																															0X22	
																															MTU_CFG	
																															0X23	
																															CC_EN	
																															0X24	
																															0X25	
																															0X26	
																															0X27	
																															0X28	
																															0X29	
																															0X2A	
																															UCNA	
																															RSVD	

图 D-1 CFG0\_BASIC 结构

表 D-1 CFG0\_BASIC Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x2C

## D.1.2 Total Number of Ports

表 D-2 Total Number of Ports 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Total Number of Ports	15:0	支持的 Port 数量，包括 physical Port 和 virtual Port。	RO	0x1	具体实现决定

## D.1.3 Total Number of Entities

表 D-3 Total Number of Entities 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Total Number of Entities	31:16	支持的 Entity 数量，包含 Entity 0。	RO_DE0_EO	0x1	具体实现决定

## D.1.4 CFG0\_CAP Bitmap

表 D-4 CFG0\_CAP Bitmap 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
CFG0_CAP Bitmap	255:0	Entity 支持的 CFG0_CAP 类型。Bit[N]的含义： 1b：CFG0_CAP_N 存在 0b：CFG0_CAP_N 不存在 注：bit0 无效。	RO	0x2	具体实现决定

## D.1.5 Supported Feature

表 D-5 支持特性寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	127:10	保留	RO	0xA	

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
CC Supported	9	指示是否支持拥塞控制 1b: 支持 0b: 不支持	RO	0xA	具体实现决定
RSVD	8	保留	RO	0xA	
Switch Supported	7	指示总线控制器是否具备转发能力： 1b: 具备 0b: 不具备 注：仅总线控制器可具备该能力。	RO	0xA	具体实现决定
RSVD	6	Reserved	-	0xA	-
UPI Supported	5	指示是否支持 UPI 1b: 支持， UPI 寄存器存在。 0b: 不支持， UPI 寄存器为 RO 属性。	RO	0xA	1'b1
Route Table Supported	4	指示配置空间是否存在 Route Table 1b: 存在 0b: 不存在 注：仅 Entity 0 具备，其余 Entity 不具备。	RO	0xA	具体实现决定
MTU Supported	3:1	指示可支持的传输层 MTU 000b: 1024 bytes 001b: 4096 bytes 010b: 8192 bytes Others: Reserved	RO	0xA	具体实现决定
Entity Available	0	指示该 Entity 是否可用 1b: 可用 0b: 不可用	RO	0xA	具体实现决定

## D.1.6 GUID

表 D-6 GUID 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Vendor ID	127:112	厂商唯一标识	RO	0xE	HwInit

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Device ID	111:96	设备标识, 由厂商自定义	RO		HwInit
Version	95:92	设备版本号, 由厂商自定义。	RO		HwInit
Type	91:88	描述设备的类型, 见 C.1 节。	RO		HwInit
Reserved	87:64	保留	RO		HwInit
Sequence Number	63:0	序列号, 由厂商自定义	RO		HwInit

### D.1.7 EID

表 D-7 EID 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
EID	127:0	该 Entity 的身份标识	RW	0x12	128'b0

### D.1.8 UBFM\_EID

表 D-8 UBFM\_EID 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UBFM EID	127:0	UBFM 实例所在节点的 EID	RW_DEN_RO	0x16	128'b0

### D.1.9 Primary CNA

表 D-9 Primary CNA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:24	保留	RO	0x1A	NA
Primary CNA	23:0	主网络地址, UB Controller 和 UB Switch 可从其任意端口接收目的地址是该 CNA 的数据包	RW_DEN_RO	0x1A	24'b0

### D.1.10 UPI

表 D-10 UPI 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UPI	14:0	UB Partition ID	RW	0x1F	15'b0

### D.1.11 Module

表 D-11 Module 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Module Vendor ID	31:16	模组厂商唯一标识，由协议归属组织统一分配	HwInit	0x20	HWIT
Module ID	15:0	模组标识，由模组厂商自定义	HwInit	0x20	HWIT

注 1：总线控制器虚拟实例无需支持配置 Module 信息。

注 2：一个 UB Controller 内不同 Entity 的 Module Vendor ID 应相同，Module ID 可不相同。

### D.1.12 DEV\_RST

表 D-12 设备复位寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
DEV_RST	0	设备级复位寄存器 1:复位 0:正常状态	RW1C_DE0_EO	0x21	1'b0

### D.1.13 MTU\_CFG

表 D-13 MTU 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
MTU_CFG	2:0	传输层 MTU 配置 000b: 1024 bytes 001b: 4096 bytes 010b: 8192 bytes Others: Reserved	RW_DEN_RO	0x23	3'b001

**D.1.14 CC\_EN**

表 D-14 拥塞控制启用寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
CC_EN	0	是否启用拥塞控制 0b: 拥塞控制不启用 1b: 拥塞控制启用 当 CFG0_BASIC.Supported feature.CC supported 为 1 时有效。	RW_DEN_RO	0x24	1'b0

**D.1.15 Trust\_EN**

表 D-15 信任 UBUU 启用寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Trust_EN	0	是否启用信任 UBUU 模式。 超节点场景下，缺省不信任模式。此寄存器仅存在于总线控制器。 0b: 不信任 UBUU，总线控制器只允许发出自身 UPI 的数据包 1b: 信任 UBUU，总线控制器可以发出携带 VM 自身 UPI 的数据包。	RW_DE0_EO	0x25	1'b0

**D.1.16 UBFM\_CNA**

表 D-16 UBFM\_CNA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UBFM CNA	23:0	UBFM 实例所在节点的 CNA	RW_DE0_EO	0x26	24'b0

### D.1.17 UEID

表 D-17 UEID 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UEID	127:0	User 的 EID	RW	0x27	128'b0

### D.1.18 UCNA

表 D-18 UCNA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UCNA	23:0	User 的 CNA	RW	0x2B	24'b0

## D.2 CFG0\_CAP

### D.2.1 CFG0\_CAP2\_SHP

#### D.2.1.1 整体结构

整体结构如下，当不支持热插拔功能时，该寄存器组可不实现。下述寄存器偏移地址计算公式中的 N 代表 Slot Index。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
Slice Header																															0x0	
Reserved																Slot Number														0x1		
Reserved																PWCS	PDSS	PLPS	WLPS	PPS										0x2		
Slot End Port Index																Slot Start Port Index														0x3		
Reserved																					PP_Ctrl									0x4		
Reserved																					WL_Ctrl									0x5		
Reserved																					PL_Ctrl									0x6		
Reserved																					MS_Ctrl									0x7		
Reserved																					PD_Ctrl									0x8		
Reserved																					PDS_Ctrl									0x9		
Reserved																					PW_Ctrl									0xA		
Reserved																					PP_Si									0xB		
Reserved																					PD_Si									0xC		
Reserved																					PDSC_Si									0xD		
Reserved																														0xE		
Reserved																														0xF		

图 D-2 SHP 整体结构

表 D-19 SHP Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	基于 Slot Number 填写

### D.2.1.2 Slot Number

表 D-20 Slot Number 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Slot Number	15: 0	支持的插槽数量，仅总线控制器和 UB Switch 可支持。	HwInit_DE0_EO	0x1	HwInit

### D.2.1.3 Capability

表 D-21 SHP Capability 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:5	Reserved	-	0x2 + N*0x10	-
PWCS	4	是否支持槽位电源控制器 0b: 不支持 1b: 支持	RO_DE0_EO	0x2 + N*0x10	具体实现决定
PDSS	3	是否支持带内在位检测 0b: 不支持 1b: 支持	RO_DE0_EO	0x2 + N*0x10	具体实现决定
PLPS	2	是否支持电源状态指示灯 0b: 不支持 1b: 支持	RO_DE0_EO	0x2 + N*0x10	具体实现决定
WLPS	1	是否支持设备工作状态指示灯 0b: 不支持 1b: 支持	RO_DE0_EO	0x2 + N*0x10	具体实现决定
PPS	0	是否支持按键 0b: 不支持 1b: 支持	RO_DE0_EO	0x2 + N*0x10	具体实现决定

表 D-22 Slot Port 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Slot End Port Index	31: 16	该 Slot 对应的终止 Port Index	HwInit_DE0_EO	0x3 + N*0x10	具体实现决定
Slot Start Port Index	15: 0	该 Slot 对应的起始 Port Index	HwInit_DE0_EO	0x3 + N*0x10	具体实现决定

#### D.2.1.4 Control

表 D-23 SHP Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
PP_CTRL	0	按键控制，只有被启用的按键，当用户按下时，对应的 slot 才会在允许的情况下产生按键消息，按键状态位才能被置位。 0b: 不启用 1b: 启用	RW_DE0_EO	0x4 + N*0x10	0
WL_CTRL	1: 0	设备工作状态指示灯控制 00b: 灭灯 01b: 亮灯 10b: 闪烁 11b: 保留	RW_DE0_EO	0x5 + N*0x10	0
PL_CTRL	1: 0	电源状态指示灯控制 00b: 灭灯 01b: 亮灯 10b: 闪烁 11b: 保留	RW_DE0_EO	0x6 + N*0x10	0
MS_CTRL	0	热插拔消息发送控制,向 CFG0.UBFM 寄存器指定的目的发送热插拔消息。 0b: 不启用 1b: 启用	RW_DE0_EO	0x7 + N*0x10	0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
PD_CTRL	0	带内设备在位检测控制 0b: 不启用 1b: 启用	RW_DE0_EO	0x8 + N*0x10	0
PDS_CTRL	0	带内设备在位检测状态变化控制 0b: 不启用 1b: 启用	RW_DE0_EO	0x9 + N*0x10	0
PW_CTRL	0	槽位电源控制 0: 关闭槽位电源 1: 开启槽位电源	RW_DE0_EO	0xA + N*0x10	0

#### D.2.1.5 Status

表 D-24 SHP Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
PP_ST	0	按键状态 0b: 按键未被按 1b: 按键被按下	RW1C_DE0_EO	0xB + N*0x10	0
PD_ST	0	设备在位检测状态 0b: 设备不在位 1b: 设备在位	RO_DE0_EO	0xC + N*0x10	0
PDSC_ST	0	设备在位检测状态变化状态 0b: 设备在位状态无变化 1b: 设备在位状态发生变化	RW1C_DE0_EO	0xD + N*0x10	0

## D.2.2 CFG0\_CAP3\_DEVICE\_ERR\_RECORD

### D.2.2.1 Slice Header

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
Slice Header																													0x00			
Reserved																													0x01~0x11			
Device Uncorrectable Error Status																													0x12			
Device Uncorrectable Error Mask																													0x13			
Device Uncorrectable Error Severity																													0x14			
Device Correctable Error Status																													0x15			
Device Correctable Error Mask																													0x16			
Device Correctable Error Severity																													0x17			
Reserved																													0x18			
Reserved																													0x19			
Device Uncorrectable Error Status																													0x1A			
Device Uncorrectable Error Mask																													0x1B			
Device Uncorrectable Error Severity																													0x1C			
Device Correctable Error Status																													0x1D			
Reserved																													0x1E			

图 D-3 DEVICE ERR RECORD Slice 结构

表 D-25 DEVICE ERR RECORD Slice Header

Field Name	Value
Slice Version	0x00
Slice Used Size	0x1E

发生设备级错误时，将 Error 状态记录在 DEVICE ERR RECORD Register 中。

注：DEVICE ERR RECORD 与 PORT ERR RECORD 的错误状态寄存器保持一致，但并非所有的错误都会在 DEVICE ERR RECORD 上报，具体在哪个寄存器上报详见 10.6.2.2.4 节 C 类错误的上报描述。

### D.2.2.2 Device Uncorrectable Error Status

表 D-26 Device Uncorrectable Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	19:0	预留给 20 种可纠正错误升级为 NFUE 错误的位置。	RO	0x12	0x0
DL Protocol Error Status	20	记录接收 ACK 偏差超过最大值的错误。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x12	0x0
DL Replay Abort Error Status	21	重传过程发生异常，记录重传机制无法恢复的错误。 此错误被配置为 NFUE 错	RW1CS_DE0_EO	0x12	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		误时，此寄存器才生效。			
RSVD	22	Reserved	RO	0x12	0x0
Packet Length Error Status	23	记录接收数据包长度检查异常的错误。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x12	0x0
RSVD	24	Reserved	RO	0x12	0x0
RSVD	25	Reserved	RO	0x12	0x0
RSVD	26	Reserved	RO	0x12	0x0
ICRC Check Error Status	27	记录接收方向 ICRC 检查异常的错误。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x12	0x0
Receiver Overflow Status	28	记录接收 Credit Buf Overflow 的错误。	RW1CS_DE0_EO	0x12	0x0
Flow Control Overflow Status	29	记录发送侧流控溢出的错误。	RW1CS_DE0_EO	0x12	0x0
RSVD	31:30	Reserved	RO	0x12	0x0

### D.2.2.3 Device Uncorrectable Error Mask

表 D-27 Device Uncorrectable Error Mask 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	19:0	预留给 20 种可纠正错误升级为 NFUE 错误的位置。	RO	0x14	0x0
DL Protocol Error Mask	20	接收 ACK 偏差超过最大值的错误屏蔽寄存器。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW_DE0_EO	0x14	0x0
DL Replay Abort Error Mask	21	重传过程发生异常，重传机制无法恢复的错误屏蔽寄存器。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW_DE0_EO	0x14	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	22	Reserved	RO	0x14	0x0
Packet Length Error Mask	23	接收数据包长度检查异常的错误屏蔽寄存器。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW_DE0_EO	0x14	0x0
RSVD	24	Reserved	RO	0x14	0x0
RSVD	25	Reserved	RO	0x14	0x0
RSVD	26	Reserved	RO	0x14	0x0
ICRC Check Error Mask	27	接收方向 ICRC 检查异常的错误屏蔽寄存器。 此错误被配置为 NFUE 错误时，此寄存器才生效。	RW_DE0_EO	0x14	0x0
Receiver Overflow Mask	28	接收 Credit Buf Overflow 的错误屏蔽寄存器。	RW_DE0_EO	0x14	0x0
Flow Control Overflow Mask	29	发送侧流控溢出的错误屏蔽寄存器。	RW_DE0_EO	0x14	0x0
RSVD	31:30	Reserved	RO	0x14	0x0

#### D.2.2.4 Device Uncorrectable Error Severity

表 D-28 Device Uncorrectable Error Severity 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Uncorrectable Error Severity	63:0	控制 Device Uncorrectable Error Status 中每 bit 错误上报 Fatal Uncorrectable Error, 还是 Non-Fatal Uncorrectable Error。 1b: 上报 Fatal Uncorrectable Error; 0b: 上报 Non-Fatal Uncorrectable Error; 仅 Bit[29:28]有效，其它 bit 不允许配置。 Bit [27:20]对应指示的错误从 CE 升级为 NFUE 时，由	RW_DE0_EO	0x16	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		Correctable Error Severity 寄存器进行配置。			

### D.2.2.5 Device Correctable Error Status

表 D-29 Device Correctable Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	0	Reserved	RO	0x18	0x0
Block Align Unlock	1	Block 失去锁定。	RW1CS_DE0_EO	0x18	0x0
Elasticity Buffer Overflow/Underflow	2	弹性 Buffer 上溢出或下溢出。	RW1CS_DE0_EO	0x18	0x0
Loss of Lane-to-Lane Deskew	3	Lane 间的 Deskew 失锁。	RW1CS_DE0_EO	0x18	0x0
AMCTL decode Error	4	AMCTL 解码错误。	RW1CS_DE0_EO	0x18	0x0
LTB CRC Error	5	LTB CRC 校验错误。	RW1CS_DE0_EO	0x18	0x0
Link speed reduce Error	6	出现非预期的降速事件。	RW1CS_DE0_EO	0x18	0x0
Link width reduce Error	7	出现非预期的降 Lane 事件。	RW1CS_DE0_EO	0x18	0x0
Equalization Coarsestune Timeout	8	Equalization 粗调阶段超时。	RW1CS_DE0_EO	0x18	0x0
Equalization Finetune Timeout	9	EQ.Passive 状态或 EQ.Active 状态超时	RW1CS_DE0_EO	0x18	0x0
Link width negotiation Timeout	10	Link width 协商超时。	RW1CS_DE0_EO	0x18	0x0
LMSM Discovery Timeout	11	LMSM Discovery 阶段超时。	RW1CS_DE0_EO	0x18	0x0
RSVD	13:12	Reserved	RO	0x18	0x0
QDLWS timeout	14	动态升降 Lane 超时。	RW1CS_DE0_EO	0x18	0x0
Unsupported LW Switch REQ	15	收到不支持的 Link width 切换请求。	RW1CS_DE0_EO	0x18	0x0
RSVD	16	Reserved	RO	0x18	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
LMSM Retrain state Timeout	17	LMSM Retrain state 超时。	RW1CS_DE0_EO	0x18	0x0
DL Retry ACK Timeout	18	DL 发送端等待 Retry ACK 超时。	RW1CS_DE0_EO	0x18	0x0
DL Retry Rollover	19	DL 重传相同 RcvPtr 的 Packet 超次。	RW1CS_DE0_EO	0x18	0x0
DL Protocol Error	20	数据链路层协议错误。 此错误被配置为 CE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x18	0x0
DL Retry Error	21	DL 重传状态机进入 Error 状态。 此错误被配置为 CE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x18	0x0
RSVD	22	Reserved	RO	0x18	0x0
Packet Length Error	23	数据包的长度检查出错。 此错误被配置为 CE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x18	0x0
RSVD	26:24	Reserved	RO	0x18	0x0
ICRC Check Error	27	接收方向 ICRC 校验错误。 此错误被配置为 CE 错误时，此寄存器才生效。	RW1CS_DE0_EO	0x18	0x0
RSVD	31:28	Reserved	RO	0x18	0x0

**D.2.2.6 Device Correctable Error Mask**

表 D-30 Device Correctable Error Mask 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	0	Reserved	RO	0x1A	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Block Align Unlock Mask	1	Block 失去锁定错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Elasticity Buffer Overflow/Underflow Mask	2	弹性 Buffer 上溢出或下溢出错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Loss of Lane-to-Lane Deskew Mask	3	Lane 间的 Deskew 失锁错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
AMCTL decode Error Mask	4	AMCTL 解码错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
LTB CRC Error Mask	5	LTB CRC 校验错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Link speed reduce Error Mask	6	出现非预期的降速事件错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Link width reduce Error Mask	7	出现非预期的降 Lane 事件错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Equalization Coarsestune Timeout Mask	8	Equalization 粗调阶段超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Equalization Finetune Timeout Mask	9	EQ.Passive 状态或 EQ.Active 状态超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Link width negotiation Timeout Mask	10	Link 宽度协商超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
LMSM Discovery State Timeout Mask	11	链路发现阶段超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
RSVD	12	Reserved	RO	0x1A	0x0
RSVD	13	Reserved	RO	0x1A	0x0
QDLWS timeout Mask	14	动态升降 lane 超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
Unsupport LW Switch REQ Mask	15	收到不支持的 Lane 宽切换请求错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
RSVD	16	Reserved	RO	0x1A	0x0
LMSM Retrain state Timeout Mask	17	LMSM Retrain state 超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0
DL Retry ACK Timeout Mask	18	DL 发送端等待 Retry ACK 超时错误屏蔽寄存器。	RW1CS_D E0_EO	0x1A	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
DL Retry Rollover Mask	19	DL 重传相同 RcvPtr 的 Packet 超次错误屏蔽寄存器	RW1CS_D_E0_EO	0x1A	0x0
DL Protocol Error Mask	20	数据链路层协议错误屏蔽寄存器。 此错误被配置为 CE 错误时，此寄存器才生效。	RW_DE0_EO	0x1A	0x1
DL Retry Error Mask	21	重传过程发生异常，重传机制无法恢复错误屏蔽寄存器。 此错误被配置为 CE 错误时，此寄存器才生效。	RW_DE0_EO	0x1A	0x1
RSVD	22	Reserved	RO	0x1A	0x0
Packet Length Error Mask	23	数据包的长度检查出错错误屏蔽寄存器。 此错误被配置为 CE 错误时，此寄存器才生效。	RW_DE0_EO	0x1A	0x0
RSVD	24	Reserved	RO	0x1A	0x0
RSVD	25	Reserved	RO	0x1A	0x0
RSVD	26	Reserved	RO	0x1A	0x0
ICRC Check Error Mask	27	接收方向 ICRC 检查错误屏蔽寄存器。 此错误被配置为 CE 错误时，此寄存器才生效。	RW_DE0_EO	0x1A	0x0
RSVD	31:28	Reserved	RO	0x1A	0x0

#### D.2.2.7 Device Correctable Error Severity

表 D-31 Correctable Error Severity 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Severity	31:0	控制 Correctable Status 中每 bit 错误上报 Correctable Error, 还是 Non-Fatal Uncorrectable Error。 1b: 上报 Correctable Error; 0b: 上报 Non-Fatal	RW_DE0_EO	0x1C	0xFFFFFFFF

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		Uncorrectable Error; 当前仅 Bit[27:20]生效，其它 Bit 不允许配置。			

## D.2.3 CFG0\_CAP4\_DEVICE\_ERR\_INFO

### D.2.3.1 Slice Header

表 D-32 DEVICE\_ERR\_INFO Slice Header

Field Name	Value
Slice Version	0x00
Slice Used Size	0x31

发生故障时，可以将错误信息记录在 DEVICE\_ERR\_INFO 寄存器中。在该寄存器中，地址 0x10 之前的寄存器为公共寄存器，0x10~0x31 的寄存器是错误信息队列记录的错误。其整体结构如下：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address													
Slice Header																													0x0																
Reserved																													0x1~0xF																
Error Information Control Register																													0x10																
Reserved																														0x11															
Reserved															Error Information Consumer Pointer														0x12																
Reserved															Error Information Producer Pointer														0x13																
Spec Defined Error Information																															0x14														
...																															0x2F														
Vendor Defined Error Information																															0x30														
																															0x31														

图 D-4 DEVICE\_ERR\_INFO 整体结构

### D.2.3.2 Error Information Control Register

表 D-33 Error Information Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Error Information Overflow	0	记录当前 Error Information 是否发生了溢出。	RO	0x10	0x0
RSVD	15:1	Reserved	RO	0x10	0x0
Error Information Queue Depth	31:16	当前 Error Information 的存储深度。取值为 0 时代表无 Error Information 提供。	RO	0x10	0x0

### D.2.3.3 Error Information Consumer Pointer

表 D-34 Error Information Consumer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Error Information Consumer Pointer	0	记录当前 Error Information 的消费指针，消费指针由软件进行控制。	RW_DEO_EO	0x12	0x0

### D.2.3.4 Error Information Producer Pointer

表 D-35 Error Information Producer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Error Information Producer Pointer	0	记录当前 Error Information 的生产指针，生产指针由硬件进行控制。	RO	0x13	0x0

### D.2.3.5 Spec Defined Error Information

表 D-36 Spec Defined Error Information 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Spec Defined Error Information [63:0]	63:0	Bit[63]: 1'b0:端口级错误； 1'b1:设备级错误。 Bit[62:32]: Reserved Bit[31:0]: 此异常数据包的伴随错误信息，与前述 Correctable Error Status 或 NFUE Status 保持一样的编码策略。	RO	0x14	0x0
Spec Defined Error Information [895:64]	895:64	此异常数据包的包头信息，从 LPH 起始到包头的最后一个 Byte。 其中 LPH 可能会出现不准确性，仅做参考。	RO	0x16	0x0

### D.2.3.6 Vendor Defined Error Information

表 D-37 Vendor Defined Error Information 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Vendor Defined Error Information	63:0	厂商自定义的异常信息	RO	0x30	0x0

## D.2.4 CFG0\_CAP5\_EMQ

### D.2.4.1 Slice Header

仅总线控制器存在 EMQ 寄存器。

表 D-38 EMQ Slice Header

Field Name	Value
Slice Version	0x00
Slice Used Size	0x1B

错误消息队列记录用于快速定位错误源，错误汇聚节点包括一组记录错误的控制寄存器和三组异常队列信息，其组成形式按照以下形式进行组织：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
Slice Header																													0x0			
																													0x1			
Error Message Queue Control																														0x2		
EMQ Correctable Error Control																															0x3	
EMQ Correctable Error Status																															0x4	
EMQ Non-Fatal Uncorrectable Error Control																															0x5	
EMQ Non-Fatal Uncorrectable Error Status																															0x6	
EMQ Fatal Uncorrectable Error Control																															0x7	
EMQ Fatal Uncorrectable Error Status																															0x8	
Reserved																															0x9	
Correctable Error Producer Pointer																															0x10	
Correctable Error Consumer Pointer																															0x11	
Correctable Error Information																															0x12	
0x13																															0x13	
Reserved																															0x14	
NonFatal Uncorrectable Error Producer Pointer																															0x15	
NonFatal Uncorrectable Error Consumer Pointer																															0x16	
0x17																															0x17	
Reserved																															0x18	
Fatal Uncorrectable Error Producer Pointer																															0x19	
Fatal Uncorrectable Error Consumer Pointer																															0x1A	
Fatal Uncorrectable Error Information																															0x1B	

图 D-5 EMQ 寄存器结构

**D.2.4.2 Error Message Queue Control**

表 D-39 Error Message Queue Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	63:3	Reserved	RO	0x2	0x0
Fatal Uncorrectable Error Report Enable	2	启用 Fatal Uncorrectable Error 上报。	RW_DE0_EO	0x2	0x0
Non-Fatal Uncorrectable Error Report Enable	1	启用 Non-Fatal Uncorrectable Error 上报。	RW_DE0_EO	0x2	0x0
Correctable Error Report Enable	0	启用 Correctable Error 上报。	RW_DE0_EO	0x2	0x0

**D.2.4.3 EMQ Correctable Error Control**

表 D-40 EMQ Correctable Error Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Queue Depth	31:16	Correctable Error Message 队列的深度	RO_DE0_EO	0x4	具体实现决定
RSVD	15:1	Reserved	RO_DE0_EO	0x4	0x0
Correctable Error Mask	0	Aggregation Error Status 中错误上报屏蔽开关。 1b: 屏蔽, 不上报错误; 0b: 不屏蔽, 上报错误;	RW_DE0_EO	0x4	0x0

**D.2.4.4 EMQ Correctable Error Status**

表 D-41 EMQ Correctable Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:9	Reserved	RO_DE0_EO	0x5	0x0
Correctable Error Overflow	8	Correctable Error 上报溢出 1b: 溢出 0b: 未溢出	RW1CS_DE0_EO	0x5	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	7:1	Reserved	RO_DE0_EO	0x5	0x0
Correctable Error detected	0	Correctable Error 上报状态指示。 1b: 已上报 0b: 未上报	RW1CS_DE0_EO	0x5	0x0

#### D.2.4.5 EMQ Non-Fatal Uncorrectable Error Control

表 D-42 EMQ Non-Fatal Uncorrectable Error Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Non-Fatal Uncorrectable Error Queue Depth	31:16	Non-Fatal Uncorrectable Error Message 队列的深度	RO_DE0_EO	0x6	具体实现决定
RSVD	15:1	Reserved	RO_DE0_EO	0x6	0x0
Non-Fatal Uncorrectable Error Mask	0	Aggregation Non-Fatal Uncorrectable Error Status 中错误上报屏蔽开关。 1b: 屏蔽, 不上报错误; 0b: 不屏蔽, 上报错误;	RW_DE0_EO	0x6	0x0

#### D.2.4.6 EMQ Non-Fatal Uncorrectable Error Status

表 D-43 EMQ Non-Fatal Uncorrectable Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:2	Reserved	RO_DE0_EO	0x7	0x0
Non-Fatal Uncorrectable Error Overflow	8	Non-Fatal Uncorrectable Error 上报溢出。 1b: 溢出 0b: 未溢出	RW1CS_DE0_EO	0x7	0x0
Reserved	7:2	Reserved	RO_DE0_EO	0x7	0x0
Non-Fatal Uncorrectable Error detected	0	Non-Fatal Uncorrectable Error 上报状态指示。 1b: 已上报 0b: 未上报	RW1CS_DE0_EO	0x7	0x0

**D.2.4.7 EMQ Fatal Uncorrectable Error Control**

表 D-44 EMQ Fatal Uncorrectable Error Control 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Fatal Uncorrectable Error Queue Depth	31:16	Fatal Uncorrectable Error Message 队列的深度	RO_DE0_EO	0x8	具体实现决定
RSVD	31:1	Reserved	RO_DE0_EO	0x8	0x0
Fatal Uncorrectable Error Mask	0	Aggregation Fatal Uncorrectable Error Status 中错误上报屏蔽开关。 1b: 屏蔽, 不上报错误; 0b: 不屏蔽, 上报错误;	RW_DE0_EO	0x8	0x0

**D.2.4.8 EMQ Fatal Uncorrectable Error Status**

表 D-45 EMQ Fatal Uncorrectable Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:9	Reserved	RO_DE0_EO	0x9	0x0
Fatal Uncorrectable Error Overflow	8	Fatal Uncorrectable Error 上报溢出。	RW1CS_DE0_EO	0x9	0x0
RSVD	7:1	Reserved	RO_DE0_EO	0x9	0x0
Fatal Uncorrectable Error detected	0	Fatal Uncorrectable Error 上报状态指示。 1b: 已上报 0b: 未上报	RW1CS_DE0_EO	0x9	0x0

**D.2.4.9 Correctable Error Consumer Pointer**

表 D-46 Correctable Error Consumer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Consumer Pointer	15:0	Correctable Error 消费指针。由软件进行维护, 硬件仅只读。	RW_DE0_EO	0x10	0x0

#### D.2.4.10 Correctable Error Producer Pointer

表 D-47 Correctable Error Producer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Producer Pointer	15:0	Correctable Error 生产指针。由硬件维护，软件只读。	RO_DE0_EO	0x11	0x0

#### D.2.4.11 Correctable Error Information

表 D-48 Correctable Error Information 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Level	63	错误级别： 1'b0：端口级错误。 1'b1：设备级错误。	RO_DE0_EO	0x13	0x0
RSVD	62:44	Reserved	RO_DE0_EO	0x13	0x0
Correctable Error Source Port Index	43:32	错误源节点的 Port Index 信息。	RO_DE0_EO	0x13	0x0
RSVD	31:24	Reserved	RO_DE0_EO	0x12	0x0
Correctable Error Source CNA	23:0	错误源节点的 CNA 信息。	RO_DE0_EO	0x12	0x0

通过此寄存器维护了一组深度为 Correctable Error Queue Depth 的 Error message 记录数组，Correctable Error Producer Pointer 指示当前硬件写入的位置，数组内存储的 Error Message 数量通过如下公式计算：

Correctable Error Producer Pointer: PI;

Correctable Error Consumer Pointer: CI;

Correctable Error Queue Depth: Depth

Error Message Num = (PI > CI) ? (PI-CI) : (PI+Depth-CI).

此寄存器为 Correctable Error Consumer Pointer 指向的 Error Message 对应的信息。

当队列满时，上报 Correctable Error Overflow 状态指示。

**D.2.4.12 Non-Fatal Uncorrectable Error Consumer Pointer****表 D-49 Non-Fatal Uncorrectable Error Consumer Pointer 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Non-Fatal Uncorrectable Error Consumer Pointer	15:0	Non-Fatal Uncorrectable Error 消费指针。由软件进行维护，硬件仅只读。	RW_DE0_EO	0x14	0x0

**D.2.4.13 Non-Fatal Uncorrectable Error Producer Pointer****表 D-50 Non-Fatal Uncorrectable Error Producer Pointer 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Non-Fatal Uncorrectable Error Producer Pointer	15:0	Non-Fatal Uncorrectable Error 生产指针。由硬件维护，软件只读。	RO_DE0_EO	0x15	0x0

**D.2.4.14 Non-Fatal Uncorrectable Error Information****表 D-51 Non-Fatal Uncorrectable Error Information 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Non-Fatal Uncorrectable Error Level	63	错误级别： 1'b0：端口级错误。 1'b1：设备级错误。	RO_DE0_EO	0x17	0x0
RSVD	62:44	Reserved	RO_DE0_EO	0x17	0x0
Non-Fatal Uncorrectable Error Source Port Index	43:32	错误源节点的 Port Index 信息。	RO_DE0_EO	0x17	0x0
RSVD	31:24	Reserved	RO_DE0_EO	0x16	0x0
Non-Fatal Uncorrectable Error Source CNA	23:0	错误源节点的 CNA 信息。	RO_DE0_EO	0x16	0x0

通过此寄存器维护了一组深度为 Non-Fatal Uncorrectable Error Queue Depth 的 Error message 记录数组，Non-Fatal Uncorrectable Error Producer Pointer 指示当前硬件写入的位置，数组内存储的 Error Message 数量通过如下公式计算：

Non-Fatal Uncorrectable Error Producer Pointer: PI;

Non-Fatal Uncorrectable Error Consumer Pointer: CI;

Non-Fatal Uncorrectable Error Queue Depth: Depth

Error Message Num = (PI>CI) ? (PI-CI) : (PI+Depth-Cl).

此寄存器为 Non-Fatal Uncorrectable Error Consumer Pointer 指向的 Error Message 对应的信息。  
当队列发生满时，上报 Non-Fatal Uncorrectable Error Overflow。

#### D.2.4.15 Fatal Uncorrectable Error Consumer Pointer

表 D-52 Fatal Uncorrectable Error Consumer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Fatal Uncorrectable Error Consumer Pointer	15:0	Fatal Uncorrectable Error 消费指针。由软件进行维护，硬件仅只读。	RW_DE0_EO	0x18	0x0

#### D.2.4.16 Fatal Uncorrectable Error Producer Pointer

表 D-53 Fatal Uncorrectable Error Producer Pointer 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Fatal Uncorrectable Error Producer Pointer	15:0	Fatal Uncorrectable Error 消生产指针。由硬件维护，软件只读。	RO_DE0_EO	0x19	0x0

#### D.2.4.17 Fatal Uncorrectable Error Information

表 D-54 Fatal Uncorrectable Error Information 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Fatal Uncorrectable Error Level	63	错误级别： 1'b0：端口级错误。 1'b1：设备级错误。	RO_DE0_EO	0x1B	0x0
RSVD	62:44	Reserved	-	0x1B	0x0
Fatal Uncorrectable Error Source Port Index	43:32	错误源节点的 Port Index 信息。	RO_DE0_EO	0x1B	0x0
RSVD	31:24	Reserved	-	0x1A	0x0
Uncorrectable Fatal Error Source CNA	23:0	错误源节点的 CNA 信息。	RO_DE0_EO	0x1A	0x0

通过此寄存器维护了一组深度为 Fatal Uncorrectable Error Queue Depth 的 Error message 记录数组，Fatal Uncorrectable Error Producer Pointer 指示当前硬件写入的位置，数组内存储的 Error Message 数量通过如下公式计算：

Fatal Uncorrectable Error Producer Pointer: PI;

Fatal Uncorrectable Error Consumer Pointer: CI;

Fatal Uncorrectable Error Queue Depth: Depth

Error Message Num = (PI>CI) ? (PI-CI) : (PI+Depth-Cl).

此寄存器为 Fatal Uncorrectable Error Consumer Pointer 指向的 Error Message 对应的信息。

当队列发生满时，上报 Fatal Uncorrectable Error Overflow。

## D.2.5 CFG0\_CAPN

CFG0\_CAP240 到 CFG0\_CAP255 为厂商自定义的 CAP。

## D.3 CFG1\_BASIC

### D.3.1 整体结构

CFG1\_BASIC 只有一个 Slice，整体结构如下图所示：

## 附录 D 配置空间寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
Slice Header																													0X0			
CFG1_CAP bitmap																												0X1				
Supported Feature																												0X2				
ERS0_SS																												0X3				
ERS1_SS																												0X4				
ERS2_SS																												0X5				
ERS0_SA																												0X6				
ERS1_SA																												0X7				
ERS2_SA																												0X8				
ERS0_UBA																												0X9				
ERS1_UBA																												0XA				
ERS2_UBA																												0XB				
Reserved																												0XC				
ELR done																												0XD				
Reserved																												0XE				
Reserved																												0XF				
Reserved																												0X10				
ERS0_SA																												0X11				
ERS1_SA																												0X12				
ERS2_SA																												0X13				
ERS0_UBA																												0X14				
ERS1_UBA																												0X15				
ERS2_UBA																												0X16				
Reserved																												0X17				
Reserved																												0X18				
Reserved																												0X19				
Reserved																												0X1A				
Reserved																												0X1B				
ELR																												0X1C				
SYS_PGS																												0X21				
EID_UPI TAB																												0X22				
EID_UPI TEN																												0X23				
Reserved																												0X24				
Reserved																												0X25				
Class Code																												0X26				
DEV_TOKEN_ID																												0X27				
Reserved																												0X28				
CDMA_EN																												0X29				
BUS_EN																												0X2A				
ERS_ACCESS_EN																												0X2B				
Reserved																												0X2C				
Reserved																																

表 D-55 CFG1\_BASIC Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x30

### D.3.2 CFG1\_CAP Bitmap

表 D-56 CFG1\_CAP Bitmap 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
CFG1_CAP Bitmap	255:0	CFG1 中包含的 CAP 类型。Bit[N]的含义： 1b: CFG1_CAP_N 存在 0b: CFG1_CAP_N 不存在 注: bit0 无效。	RO	0x1	具体实现决定

### D.3.3 Supported Feature

表 D-57 支持特性

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
RSVD	127:11	Reserved	RO	0x9	-
Decoder MATT jurisdiction	10	指示 Decoder 内存地址转换表的管辖权归属 0: UB 驱动 1: UBFM 注: 仅总线控制器具备该能力	HwInit	0x9	HwInit
RSVD	9	保留	RO	0x9	-
ERS2S	8	支持 ERS2 空间。 1: 支持 0: 不支持 注: 当不支持时, 配置该资源空间的寄存器将不生效。	RO	0x9	具体实现决定
ERS1S	7	支持 ERS1 空间。 1: 支持 0: 不支持 注: 当不支持时, 配置该资源空间的寄存器将不生效。	RO	0x9	具体实现决定

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
ERS0S	6	支持 ERS0 空间。 1: 支持 0: 不支持 注: 当不支持时, 配置该资源空间的寄存器将不生效。	RO	0x9	具体实现决定
UBA_Support	5	是否支持 UBA 配置寄存器。 1: 支持 0: 不支持 注: 不支持时, 配置 ERS0_UBA、ERS1_UBA、ERS2_UBA 不生效。	RO	0x9	具体实现决定
RSVD	4	保留	RO	0x9	-
RSVD	3	保留	RO	0x9	-
MGS (Migration Supported)	2	是否支持迁移 1: 支持 0: 不支持	RO	0x9	具体实现决定
RSVD	1:0	Reserved	RO	0x9	具体实现决定

### D.3.4 ERS0\_SS

表 D-58 ERS0\_SS 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
ERS0 Space Size	31:0	ERS0 地址空间大小, 单位为 SYS_PGS	RO	0xD	具体实现决定

### D.3.5 ERS1\_SS

表 D-59 ERS1\_SS 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
ERS1 Space Size	31:0	ERS1 地址空间大小, 单位为 SYS_PGS	RO	0xE	具体实现决定

### D.3.6 ERS2\_SS

表 D-60 ERS2\_SS 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS2 Space Size	31:0	ERS2 地址空间大小，单位为 SYS_PGS	RO	0xF	具体实现决定

### D.3.7 ERS0\_SA

表 D-61 ERS0\_SA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS0 Start Address	63:0	ERS0 空间起始地址，设备内部地址偏移	RO	0x10	具体实现决定

### D.3.8 ERS1\_SA

表 D-62 ERS1\_SA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS1 Start Address	63:0	ERS1 空间起始地址，设备内部地址偏移	RO	0x12	具体实现决定

### D.3.9 ERS2\_SA

表 D-63 ERS2\_SA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS2 Start Address	63:0	ERS2 空间起始地址，设备内部地址偏移	RO	0x14	具体实现决定

### D.3.10 ERS0\_UBA

表 D-64 ERS0\_UBA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS0 UBA	63:0	ERS0 的起始 UBA 地址 注 1：仅当 UBAS 为 1 时，该寄存器存在 注 2：Entity n 的该寄存器属性为 RO，其值由 Entity 0 的 ERS0_UBA + 本 Entity	RW_DEN_RO	0x16	具体实现决定

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
		的 ERS0_SA 计算得到（软件读取值为计算后的值）。			

### D.3.11 ERS1\_UBA

表 D-65 ERS1\_UBA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS1_UBA	63:0	ERS1 的起始 UBA 地址 注 1：仅当 UBAS 为 1 时，该寄存器存在； 注 2：Entity n 的该寄存器属性为 RO，其值由 Entity 0 的 ERS1_UBA + 本 Entity 的 ERS1_SA 计算得到（软件读取值为计算后的值）。	RW_DEN_RO	0x18	具体实现决定

### D.3.12 ERS2\_UBA

表 D-66 ERS2\_UBA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ERS2_UBA	63:0	ERS2 的起始 UBA 地址 注 1：仅当 UBAS 为 1 时，该寄存器存在； 注 2：Entity n 的该寄存器属性为 RO，其值由 Entity 0 的 ERS2_UBA + 本 Entity 的 ERS2_SA 计算得到（软件读取值为计算后的值）。	RW_DEN_RO	0x1A	具体实现决定

### D.3.13 ELR

表 D-67 ELR 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
ELR	0	复位控制： 1：复位	RW1C	0x1C	1'b0

**D.3.14 ELR done****表 D-68 ELR done 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
ELR done	0	复位完成状态指示： 0：正常状态或复位未完成状态 1：复位完成状态	RO	0x1D	1'b0

**D.3.15 SYS\_PGS****表 D-69 SYS\_PGS 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
System page granule size	0	System page granule size 大小： 0：4K bytes 1：64K bytes	RW_DEN_RO	0x21	1'b0

**D.3.16 EID\_UPI\_TAB****表 D-70 EID\_UPI\_TAB 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
EID_UPI Table Base Address	63:0	EID_UPI Table 表基地址寄存器，用于指示 EID_UPI 表的存放位置。 此寄存器仅存在于总线控制器中。	RW_DE0_EO	0x22	64'b0

在具体实现上，EID\_UPI 表可以保存在 UBPU 内存或总线控制器片上 RAM 内。当保存在 UBPU 内存时，UBPU 应根据 EID\_UPI TEN 寄存器的指示分配一段虚拟地址连续的内存，并将内存首地址设置到 EID\_UPI TBA 寄存器中。当保存在片上 RAM 时，可提供更高的安全级别，具体实现不在本规范定义。启动虚机后应将此虚机对应的(EID, UPI)写入到 EID\_UPI 表中。

**D.3.17 EID\_UPI\_TEN**

表 D-71 EID\_UPI\_TEN 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
EID_UPI Table Entry Num	31:0	指示 EID_UPI Table 需要多少个 Entry。 此寄存器仅存在于总线控制器中。	RO	0x24	具体实现决定

**D.3.18 Class Code**

表 D-72 Class Code 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
Class Code	15:0	指示 Entity 声明的功能或服务，见 C.2 节。	RO	0x29	具体实现决定

**D.3.19 DEV\_TOKEN\_ID**

表 D-73 DEV\_TOKEN\_ID 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
DEV_TOKEN_ID	19:0	OS 分配的 TokenID	RW	0x2D	1'b0

**D.3.20 BUS\_EN**

表 D-74 BUS\_EN 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
bus_access_en	0	发起访问的启用开关 0: 不启用 1: 启用	RW	0x2E	1'b0

**D.3.21 ERS\_ACCESS\_EN**

表 D-75 ERS\_ACCESS\_EN 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
dev_rs_access_en	0	访问资源空间的启用开关	RW	0x2F	1'b0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		0: 不启用 1: 启用			

## D.4 CFG1\_CAP

### D.4.1 CFG1\_CAP1\_DECODER

#### D.4.1.1 Slice Header

表 D-76 CFG1\_CAP1\_DECODER Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x25

#### D.4.1.2 DECODER

表 D-77 DECODER 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:19	保留	RO_DE0_EO	0x1	-
mmio_size	18:16	MMIO SIZE 支持情况: 000b: 128Gbyte 001b: 256Gbyte 010b: 512Gbyte 011b: 1Tbyte 100b: 2Tbyte 101b: 4Tbyte 110b: 8Tbyte 111b: 16Tbyte	RO_DE0_EO	0x1	具体实现决定
Cmdq_depth	15:12	CMD Queue 的深度	RO_DE0_EO	0x1	具体实现决定
RSVD	11:8	保留	RO_DE0_EO	0x1	-
Eventq_depth	7:4	Event Queue 的深度	RO_DE0_EO	0x1	具体实现决定
RSVD	3:0	保留	RO_DE0_EO	0x1	-

**D.4.1.3 DECODER\_CTRL****表 D-78 DECODER\_CTRL 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:1	保留	RO_DE0_EO	0x2	1'b0
Decoder_en	0	Decoder 启用控制: 0: Disable 1: Enable	RW_DE0_EO	0x2	1'b0

**D.4.1.4 DECODER\_MATT\_BA****表 D-79 DECODER\_MATT\_BA 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
DECODER_MATT_BA	63:0	Decoder 内存地址转换表的入口基地址。  注：该寄存器仅总线控制器存在。	RW_DE0_EO	0x3	64'b0

**D.4.1.5 DECODER\_MMIO\_BA****表 D-80 DECODER\_MMIO\_BA 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
DECODER_MMIO_BA	63:0	Decoder 对应的 MMIO 访问空间基地址  Decoder_MMIO_BASE_ADDR ( Decoder 地址转换地址 = 输入 HPA- Decoder_MMIO_BASE_ADDR )  注：该寄存器仅总线控制器存在。	RW_DE0_EO	0x5	64'b0

**D.4.1.6 DECODER\_USI\_IDX****表 D-81 DECODER\_USI\_IDX 寄存器**

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
DECODER_USI_IDX	31:0	指示 Decoder 对应的 event 中断上报使用的 USI 中断向量号。  注：该寄存器仅总线控制器存在。	RO_DE0_EO	0x7	具体实现决定

**D.4.1.7 DECODER\_CMDQ\_CFG****表 D-82 DECODER\_CMDQ\_CFG 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Value</b>
RSVD	31:12	Reserved	RO_DE0_EO	0x10	-
cmdq_depth_use	11:8	CMDQ 深度由软件配置，深度为 $2^{\text{cmdq\_depth\_use}}$ 要求：cmdq_depth_use > 0	RW_DE0_EO	0x10	具体实现决定
RSVD	7:1	Reserved	RO_DE0_EO	0x10	-
cmdq_en	0	CMDQ 启用控制： 0: 不启用 1: 启用	RW_DE0_EO	0x10	1'b0

**D.4.1.8 DECODER\_CMDQ\_PROD****表 D-83 DECODER\_CMDQ\_PROD 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Value</b>
RSVD	31:17	Reserved	RO_DE0_EO	0x11	-
cmdq_err_resp	16	CQ 错误 flag	RW_DE0_EO	0x11	1'b0
RSVD	15:11	Reserved	RO_DE0_EO	0x11	-
cmdq_wr_idx	10:0	CQ 写索引  Command queue write index wr_idx[10:QS+1] is reserved;(if QS < 10) wr_idx[QS] is write index wrap flag; wr_idx[QS-1:0] is write index; 说明：“QS”是指 DECODER_CMDQ_CFG.cmdq_size_use,注意要求 “cmdq_size_use”大于 0.	RW_DE0_EO	0x11	11'b0

**D.4.1.9 DECODER\_CMDQ\_CONS****表 D-84 DECODER\_CMDQ\_CONS 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Value</b>
RSVD	31:20	Reserved	RO_DE0_EO	0x12	-

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
cmdq_err_reason	19: 17	指示 CMDQ 错误类型： 3'b000: 无错误 3'b001: 非法命令 3'b010: abort error (read command with 2bit ecc) Others: Reserved 注: 当 DECODER_CMDQ_CFG.cmdq_en 值为 0 时,该字段属性为 RW,否则为 RO	RW_DE0_EO	0x12	3'b0
cmdq_err	16	CMDQ 错误指示 1: 存在错误 注: 当 DECODER_CMDQ_CFG.cmdq_en 值为 0 时,该字段属性为 RW,否则为 RO	RW_DE0_EO	0x12	1'b0
RSVD	15:11	Reserved	RO_DE0_EO	0x12	-
cmdq_rd_idx	10:0	CMDQ 读索引 rd_idx[10:QS+1] is reserved;(if QS < 10) rd_idx[QS] is read index wrap flag; rd_idx[QS-1:0] is read index; 说明: “QS”是指 DECODER_CMDQ_CFG.cmdq_size_use,注意要求“cmdq_size_use”大于 0. 注: 当 DECODER_CMDQ_CFG.cmdq_en 值为 0 时,该字段属性为 RW,否则为 RO	RW_DE0_EO	0x12	11'b0

#### D.4.1.10 DECODER\_CMDQ\_BASE\_ADDR

表 D-85 DECODER\_CMDQ\_BASE\_ADDR 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	63:48	Reserved	RO_DE0_EO	0x13	-
cmdq_base_addr	47:6	CMDQ 队列起始地址的[47:6] 注: 当 DECODER_CMDQ_CFG.cmdq_en 值为 0 时,该字段属性为 RW,否则为 RO	RW_DE0_EO		42'b0
RSVD	5:0	Reserved	RO_DE0_EO		

## D.4.1.11 DECODER\_EVENTQ\_CFG

表 D-86 DECODER\_EVENTQ\_CFG 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:12	Reserved	RO_DE0_EO	0x20	-
eventq_depth_use	11:8	Event queue 深度由软件配置，队列深度为 $2^{\text{eventq\_depth\_use}}$ <b>要求:</b> eventq_depth_use > 0	RW_DE0_EO	0x20	4'b0
RSVD	7:1	Reserved	RO_DE0_EO	0x20	-
eventq_en	0	Event queue 启用控制: 0: 不启用 1: 启用	RW_DE0_EO	0x20	1'b0

## D.4.1.12 DECODER\_EVENTQ\_PROD

表 D-87 DECODER\_EVENTQ\_PROD 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
eventq_ovfl_err	31	指示 Event queue 产生了 Overflow 错误 注:当 DECODER_EVENTQ_CFG.eventq_en 为 0 时,该域段属性为 RW,否则为 RO	RO_DE0_EO	0x21	1'b0
RSVD	30:11	Reserved	RO_DE0_EO	0x21	-
eventq_wr_idx	10:0	Event queue write index wr_idx[10:QS+1] is reserved;(if QS < 10) wr_idx[QS] is write index wrap flag; wr_idx[QS-1:0] is write index; <b>说明:</b> “QS”是指 DECODER_EVENTQ_CFG.eventq_size_use,注意要求“eventq_size_use”大于 0. 注:当 DECODER_EVENTQ_CFG.eventq_en 为 0 时,该域段属性为 RW,否则为 RO	RW_DE0_EO	0x21	11'b0

## D.4.1.13 DECODER\_EVENTQ\_CONS

表 D-88 DECODER\_EVENTQ\_CONS 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
eventq_ovfl_err_resp	31	Event queue Overflow acknowledge flag	RO_DE0_EO	0x22	1'b0
RSVD	30:11	Reserved	-	0x22	-
eventq_rd_idx	10:0	<p>Event queue write index  rd_idx[10:QS+1] is reserved;(if QS &lt; 10)  rd_idx[QS] is write index wrap flag;  rd_idx[QS-1:0] is write index;</p> <p><b>说明:</b> “QS”是指  DECODER_EVENTQ_CF  G.eventq_size_use,注意  要求“eventq_size_use”大  于0.</p> <p><b>Note:</b>When  DECODER_EVENTQ_CF  G.event_en==0,access to  this field is RW;  otherwise access to this  field is RO</p>	RW_DE0_EO	0x22	11'b0

## D.4.1.14 DECODER\_EVENTQ\_BASE\_ADDR

表 D-89 DECODER\_EVENTQ\_BASE\_ADDR 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	63:48	Reserved	-	0x23	-
eventq_base_addr	47:6	<p>Event queue 起始地址的 [47:6]</p> <p>注:当  DECODER_EVENTQ_CFG.  eventq_en 为 0 时,该域段属  性为 RW,否则为 RO</p>	RW_DE0_EO		42'b0
RSVD	5:0	Reserved	-		

**D.4.2 CFG1\_CAP2\_JETTY**

表 D-90 Jetty Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x2

表 D-91 Jetty 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Event Queue Number	31:16	支持的 EQ 数量，单位为个。	RO	0x1	具体实现决定
RSVD	15:12	保留	RO	0x1	具体实现决定
JFR Max Number	11:8	支持的 JFR 数量为 $8^*2^k$ , k 为该域段值	RO	0x1	具体实现决定
JFC Max Number	7:4	支持的 JFC 数量为 $8^*2^m$ , m 为该域段值	RO	0x1	具体实现决定
JFS and Jetty Max Number	3:0	支持的 JFS/Jetty 数量为 $8^*2^n$ , n 为该域段值	RO	0x1	具体实现决定

**D.4.3 CFG1\_CAP3\_INT\_TYPE1**

实现 Type1 中断时，应支持该寄存器组。总线控制器应使用这种方式。

表 D-92 INT\_TYPE1 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

表 D-93 INT\_TYPE1 寄存器组

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Interrupt Enable	0	所有中断启用的总开关 1: 启用 0: 不启用	RW	0x1	1'b0
Supported Interrupt Number	2:0	支持的中断向量数量 000b: 1 个 001b: 2 个	RO	0x2	3'b0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		010b: 4 个 011b: 8 个 100b: 16 个 101b: 32 个 Others: 保留			
Interrupt_Enable_number	2:0	启用的中断向量数量 000b: 1 个 001b: 2 个 010b: 4 个 011b: 8 个 100b: 16 个 101b: 32 个 Others: 保留	RW	0x3	3'b0
Interrupt_Data	31:0	中断数据	RW	0x4	32'b0
Interrupt_address	63:0	中断地址	RW	0x5	64'b0
Interrupt ID	31:0	中断标识	RW	0x7	32'b0
Interrupt_Mask	31:0	指示对应的中断向量是否被屏蔽，32bit 分别对应 32 个中断向量编号，bit0 描述中断向量 0。 0: 不屏蔽； 1: 屏蔽；	RW	0x8	32'HFFFF_FFFF
Interrupt_Pending	31:0	指示对应的中断向量上报状态，32bit 分别对应 32 个中断向量编号，bit0 描述中断向量 0。 0: 已上报； 1: 待上报；	RO	0x9	32'b0

#### D.4.4 CFG1\_CAP4\_INT\_TYPE2

支持 Type2 中断时，应实现该寄存器组。

表 D-94 INT\_TYPE2 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xB

表 D-95 INT\_TYPE2 寄存器组

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Max Index of Interrupt Address	31:16	指示中断地址的最大索引，Entity 支持的中断地址数量为 M+1，M 为该寄存器值。	RO	0x1	具体实现决定
Max Index of Interrupt Vector	15:0	指示中断向量的最大索引，Entity 支持的中断向量数量为 N+1，N 为该寄存器值。	RO	0x1	具体实现决定
Interrupt Vector Table Start Address	63:0	指示中断向量表的起始地址	RO	0x2	具体实现决定
Interrupt Address Table Start Address	63:0	指示中断地址表的起始地址	RO	0x4	具体实现决定
Interrupt Pending Table Start Address	63:0	指示中断 Pending 表的起始地址	RO	0x6	具体实现决定
Interrupt ID	31:0	中断标识	RW	0x8	32'b0
Interrupt Mask	0	所有中断的屏蔽开关 1：屏蔽所有中断 0：不屏蔽所有中断	RW	0x9	1'b1
Interrupt Enable	0	所有中断的启用控制 1：启用 0：不启用	RW	0xA	1'b0

#### D.4.5 CFG1\_CAP6\_UB\_MEM

仅总线控制器需实现该寄存器组。

表 D-96 UB\_MEM Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x2

表 D-97 UB\_MEM 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
UB_MEM_USI_IDX	31: 0	UB Memory 特性对应的 event 中断上报使用的 USI 中断向量号。	RO_DE0_EO	0x1	具体实现决定

## D.4.6 CFG1\_CAPN

CFG1\_CAP240 到 CFG1\_CAP255 为厂商自定义的 CAP。

## D.5 CFG0\_PORT\_BASIC

### D.5.1 Slice Header

所有 Port 的 PORT\_BASIC 的寄存器偏移和含义相同，只有访问基地址不同。

表 D-98 CFG0\_PORT\_BASIC Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x11

整体结构如下：

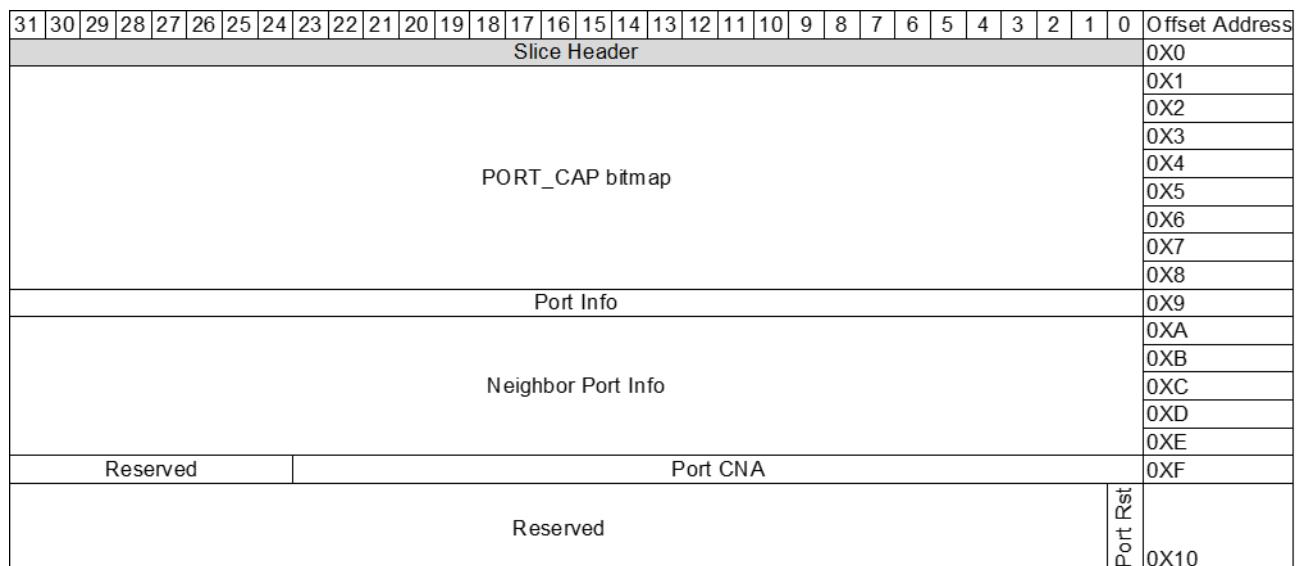


图 D-7 CFG0\_PORT\_BASIC 整体结构

### D.5.2 PORT\_CAP Bitmap

表 D-99 PORT\_CAP Bitmap 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
PORT_CAP Bitmap	255:0	Port 支持的 CAP 类型。 Bit[N]的含义： 1: PORT_CAP_N 存在 0: PORT_CAP_N 不存在 注: bit0 无效。	RO_DE0_EO	0x1	-

### D.5.3 Port Info

表 D-100 Port Info 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:18	Reserved	RO_DE0_EO	0x9	NA
Enumeration Boundary	17	0: 该 Port 不是枚举边界端口； 1: 该 Port 是枚举边界端口；	HwInit_DE0_EO	0x9	HwInit
Port type	16	Port 类型 0b: physical Port 1b: vPort vPort 仅拥有如下寄存器：Port Index、Enumeration Boundary、Port CNA、Neighbor Port Info。	RO_DE0_EO	0x9	-
Port Index	15:0	Port Index	RO_DE0_EO	0x9	-

注：仅 UB Switch 和总线控制器拥有枚举边界寄存器，其它 UB Controller 都是天然的枚举边界。UBFM 实例和系统软件都会发起枚举，但两个管理角色可枚举的范围不同。

1. 系统软件枚举范围：系统软件在总线控制器端口的 Enumeration Boundary 属性为 1 时，结束在该路径上的枚举；若 Enumeration Boundary 属性为 0，则表示未到边界，可以继续进行下一级枚举，发现私有设备。
2. UBFM 实例枚举范围：UBFM 实例在枚举到 UB Switch 后，若其某个端口的 Enumeration Boundary 属性为 0 且链路邻居是总线控制器，虽然总线控制器端口的 Enumeration Boundary 属性为 1，UBFM 实例不受该端口边界属性限制，继续枚举到该总线控制器为止。

### D.5.4 Neighbor Port Info

表 D-101 Neighbor Port Info 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
RSVD	31:16	Reserved	RO_DE0_EO	0xA	
Neighbor_port_idx	15:0	链路邻居的端口索引	RO_DE0_EO	0xA	
Neighbor_port_GUID	127: 0	链路邻居 Entity 0 的 GUID	RO_DE0_EO	0xB	

### D.5.5 Port CNA

UB Controller 的每个 Port 都有一个 CNA 地址。UB Switch 不存在本寄存器。

表 D-102 Port CNA 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
RSVD	31:24	保留	RO_DE0_EO	0xF	-
Port CNA	23:0	默认值为全 0，非全 0 值表示 CNA 已分配	RW_DE0_EO	0xF	24'b0

### D.5.6 Port Rst

表 D-103 Port Rst 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
RSVD	31:1	保留	RO_DE0_EO	0x10	NA
Port Reset	0	端口复位控制 1:复位 0:正常状态	RW1C_DE0_EO	0x10	NA

## D.6 CFG0\_PORT\_CAP

### D.6.1 一般要求

所有 Port 的 PORT\_CAP 内规范定义的寄存器偏移和含义相同，只有访问基地址不同。

## D.6.2 PORT\_CAP1\_LINK

表 D-104 LINK Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xc7

### D.6.2.1 Link\_Capability

#### 1. Data Link Capability 1

表 D-105 Data Link Capability 1 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:17	NA	RO_DE0_EO	0x1	NA
RXBUF_VL_SHA RE_pro	16	本端 RXBUF 是否支持 VL 共享 1'b1: 支持; 1'b0: 不支持	RO_DE0_EO	0x1	NA
FEATURE_ID	15:0	UB 协议版本匹配 ID	RO_DE0_EO	0x1	NA
Reserved	31:16	NA	RO_DE0_EO	0x2	NA
CTRL_ACK_GRA IN_SIZE_pro	15:8	Crd_Ack Block 返回 ACK 支持的粒度: 数据包头返 还信用证支持的粒度: 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x2	NA
DATA_ACK_GR AIN_SIZE_pro	7:0	业务包头返还 ACK 的粒 度: 数据包头返还信用证 支持的粒度: 每 bit 对应一 个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x2	NA
CTRL_CREDIT_ GRAIN_SIZE_pro _0	31:0	Crd_Ack Block 支持 VL0~VL3 的 credit 返回粒 度 (单位: CELL)  每个 byte 对应一个 VL 的 credit 返回粒度{VL3,VL2, VL1,VL0}  每 bit 对应一个 CELL 数 量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x3	NA

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
CTRL_CREDIT_GRAIN_SIZE_pro_1	31:0	Crd_Ack Block 支持 VL4~VL7 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度{VL7,VL6, VL5,VL4} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x4	NA
CTRL_CREDIT_GRAIN_SIZE_pro_2	31:0	Crd_Ack Block 支持 VL8~VL11 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL11,VL10, VL9,VL8} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x5	NA
CTRL_CREDIT_GRAIN_SIZE_pro_3	31:0	Crd_Ack Block 支持 VL12~VL15 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL15,VL14, VL13,VL12} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x6	NA
DATA_CREDIT_GRAIN_SIZE_pro_0	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL0~VL3 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL3,VL2,VL1,VL0} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x7	NA
DATA_CREDIT_GRAIN_SIZE_pro_1	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL4~VL7 的 credit	RO_DE0_EO	0x8	NA

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL7,VL6,VL5,VL4} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}			
DATA_CREDIT_GRAIN_SIZE_pro_2	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL8~VL11 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL11,VL10,VL9,VL8} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0x9	NA
DATA_CREDIT_GRAIN_SIZE_pro_3	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL12~VL15 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL15,VL14,VL13,VL12} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa	NA
PACKET_MIN_INTERVAL	31:24	最小包间隙, 单位为 Flit。	RO_DE0_EO	0xb	NA
FLOW_CTRL_SIZE_pro	23:16	支持信用流控最小单元 CELL 对应的 FLIT 数量: 每 bit 对应一个 FLIT 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xb	NA
VL_ENABLE_pro	15:0	是否支持虚通道	RO_DE0_EO	0xb	NA
Reserved	31:20	NA	RO_DE0_EO	0xc	NA
RETRY_BUF_DEPTH	19:0	重传 BUF 深度	RO_DE0_EO	0xc	NA

## 2. PHY Link Capability 1

Start Offset Address:0x20

表 D-106 PHY Link Capability 1 寄存器

Field Name	Bit Location	Description	Attribute	Default Value
Max Link Speed	3:0	端口支持的最大链路速率。 4'h0: Data Rate0 4'h1: Data Rate1 4'h2: Data Rate2 4'h3: Data Rate3 4'h4: Data Rate4 4'h5: Data Rate5 4'h6: Data Rate6 4'h7: Data Rate7 4'h8: Data Rate8 4'h9: Data Rate9 Others: Reserved	RO_DE0_EO	
Maximum Link Width	9:4	TX 和 RX 的最大链路宽度(xN – 对应 N 个 Lanes)。 6'h0: Reserved 6'h1: x1 6'h2: x2 6'h4: x4 6'h8: x8 Others: Reserved	RO_DE0_EO	
PHY Mode support	15:10	指示物理层支持的工作模式。每一位 bit 值为 1'b1 表示支持, 否则表示不支持。 Bit0: PHY-Mode1 Bit1: PHY-Mode2 Bit2-Bit5: Reserved	RO_DE0_EO	
FEC Mode support	23:16	指示物理层支持的 FEC 模式。每一位 bit 值为 1'b1 表示支持, 否则表示不支持。 Bit 0: RS (128, 120,T=2) Bit 1: RS (128, 120,T=4) Bit [7:2]: Reserved	RO_DE0_EO	
EQ Mode Support	27:24	指示物理层支持的 EQ 模式。每一位 bit 值为 1'b1 表示支持, 否则表示不支持。 Bit 0: Full_EQ Bit 1: Only_Highest_Data_Rate_EQ	RO_DE0_EO	

Field Name	Bit Location	Description	Attribute	Default Vaule
		Bit 2: Skip_EQ Bit 3: Reserved		
Over fibre support	28	指示物理层是否支持光链路模式。 1'b0: Not support 1'b1: Support	RO_DE0_EO	
Fix Data Rate mode support	29	指示物理层是否支持固定速率模式。 1'b0: Not Support 1'b1: Support	RO_DE0_EO	
Reserved	31:30	Reserved	RO_DE0_EO	

## 3. PHY Link Capability 2

Start Offset Address:0x21

表 D-107 PHY Link Capability 2 寄存器

Field Name	Bit Location	Description	Attribute	Default Vaule
Supported Link Speeds Vector	15:0	指示端口支持的链路速率，每一位 bit 值为 1'b1 表示支持相应的链路速率，否则表示不支持。 Bit0: Data Rate0 Bit1: Data Rate1 ... Bit9: Data Rate9 Bit10-Bit15: Reserved	RO_DE0_EO	
Reserved	31:16	Reserved	RO_DE0_EO	

## 4. PHY Link Capability 3

Start Offset Address:0x22

表 D-108 PHY Link Capability 3 寄存器

Field Name	Bit Location	Description	Attribute	Default Vaule
Asymmetry Link support	0	指示物理层是否支持不对称链路。 1'b1: Support 1'b0: Not Support	RO_DE0_EO	
Reserved	1	Reserved	RO_DE0_EO	

Field Name	Bit Location	Description	Attribute	Default Value
Precode Support-	3:2	指示物理层是否支持预编码。每一位 bit 值为 1'b1 表示支持，否则表示不支持。 bit 2: NRZ bit 3: PAM4	RO_DE0_EO	
Reserved	7:4	Reserved	RO_DE0_EO	
FEC Interleave Mode	15:8	指示物理层支持的 FEC 交织模式，每一位 bit 值 1'b1 表示支持，否则表示不支持。 Bit0: x1 FEC interleave Bit1: x2 FEC interleave Bit2: x4 FEC interleave Bit3: x8 FEC interleave Bit4-Bit7: Reserved	RO_DE0_EO	
Reserved	31:16	Reserved	RO_DE0_EO	

## 5. PHY Link Capability 4

Start Offset Address:0x23

表 D-109 PHY Link Capability 4 寄存器

Field Name	Bit Location	Description	Attribute	Default Value
Reserved	31:0	Reserved	RO_DE0_EO	

## 6. PHY Link Capability 5

Start Offset Address:0x24

表 D-110 PHY Link Capability 5 寄存器

Field Name	Bit Location	Description	Attribute	Default Value
Reserved	31:0	Reserved	RO_DE0_EO	

### D.6.2.2 Link Configuration

#### 1. Data Link Configuration

表 D-111 Data Link Configuration 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:17	NA	RW_DE0_EO	0x40	0x0
RXBUF_VL_	16	本端 RXBUF 是否支持 VL 共享	RW_DE0_EO	0x40	0x1

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
SHARE_cfg		1'b1: 支持; 1'b0: 不支持			
CTRL_ACK_GRAIN_SIZE_cfg	15:8	Crd_Ack Block 返回 ACK 支持的粒度：数据包头返还信用证支持的粒度：每 bit 对应一个 CELL 数量，分别为 {128,64,32,16,8,4,2,1}	RW_DE0_EO	0x40	0x01
DATA_ACK_GRAIN_SIZE_cfg	7:0	业务包头返还 ACK 的粒度：数据包头返还信用证支持的粒度：每 bit 对应一个 CELL 数量，分别为 {128,64,32,16,8,4,2,1}	RW_DE0_EO	0x40	0x20
CTRL_CREDIT_GRAIN_SIZE_cfg_0	31:0	Crd_Ack Block 支持 VL0~VL3 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度{VL3,VL2, VL1,VL0} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x41	0x0101 0101
CTRL_CREDIT_GRAIN_SIZE_cfg_1	31:0	Crd_Ack Block 支持 VL4~VL7 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度{VL7,VL6, VL5,VL4} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x42	0x0101 0101
CTRL_CREDIT_GRAIN_SIZE_cfg_2	31:0	Crd_Ack Block 支持 VL8~VL11 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度{VL11,VL10, VL9,VL8} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x43	0x0101 0101
CTRL_CREDIT_GRAIN_SIZE_cfg_3	31:0	Crd_Ack Block 支持 VL12~VL15 的 credit 返还粒度 (单位: CELL)	RW_DE0_EO	0x44	0x0101 0101

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		每个 byte 对应一个 VL 的 credit 返还粒度{VL15,VL14, VL13,VL12} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}			
DATA_CRED_IT_GRAIN_SIZE_cfg_0	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL0~VL3 的 credit 返还粒度（单位：CELL） 每个 byte 对应一个 VL 的 credit 返还粒度{VL3,VL2, VL1,VL0} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x45	0x0404 0404
DATA_CRED_IT_GRAIN_SIZE_cfg_1	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL4~VL7 的 credit 返还粒度（单位：CELL） 每个 byte 对应一个 VL 的 credit 返还粒度{VL7,VL6, VL5,VL4} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x46	0x0404 0404
DATA_CRED_IT_GRAIN_SIZE_cfg_2	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL8~VL11 的 credit 返还粒度（单位：CELL） 每个 byte 对应一个 VL 的 credit 返还粒度{VL11,VL10, VL9,VL8} 每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x47	0x0404 0404
DATA_CRED_IT_GRAIN_SIZE_cfg_3	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL12~VL15 的 credit 返还粒度（单位：CELL） 每个 byte 对应一个 VL 的 credit 返还粒度{VL15,VL14, VL13,VL12}	RW_DE0_EO	0x48	0x0404 0404

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
		每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}			
PACKET_MI_N_INTERVAL	31:24	支持最小包间隔，单位:FLIT	RW_DE0_EO	0x49	0x1
FLOW_CTRL_SIZE_cfg	23:16	支持信用流控最小单元 CELL 对应的 FLIT 数量：每 bit 对应一个 FLIT 数量，分别为{128,64,32,16,8,4,2,1}	RW_DE0_EO	0x49	0x08
VL_ENABLE_cfg	15:0	是否支持虚通道	RW_DE0_EO	0x49	0x00FF
Reserved	31:20	NA	RW_DE0_EO	0x4a	0x0
RETRY_BUF_DEPTH	19:0	重传 BUF 深度	RW_DE0_EO	0x4a	0x0
Reserved	31:16	NA	RW_DE0_EO	0x4b	0x0
FEATURE_ID_cfg	15:0	NA	RW_DE0_EO	0x4b	0x0001

## 2. Data Link Remote Init Credit Cfg

表 D-112 Data Link Remote Init Credit Cfg 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
Cfg Remote crd vl15	255:240	分配给对端 VL15 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x67	0x0
Cfg Remote crd vl14	239:224	分配给对端 VL14 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x67	0x0
Cfg Remote crd vl13	223:208	分配给对端 VL13 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x66	0x0
Cfg Remote crd vl12	207:192	分配给对端 VL12 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x66	0x0
Cfg Remote crd vl11	191:176	分配给对端 VL11 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x65	0x0
Cfg Remote crd vl10	175:160	分配给对端 VL10 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x65	0x0
Cfg Remote crd vl9	159:144	分配给对端 VL9 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x64	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Cfg Remote crd vl8	143:128	分配给对端 VL8 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x64	0x0
Cfg Remote crd vl7	127:112	分配给对端 VL7 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x63	0x0
Cfg Remote crd vl6	111:96	分配给对端 VL6 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x63	0x0
Cfg Remote crd vl5	95:80	分配给对端 VL5 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x62	0x0
Cfg Remote crd vl4	79:64	分配给对端 VL4 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x62	0x0
Cfg Remote crd vl3	63:48	分配给对端 VL3 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x61	0x0
Cfg Remote crd vl2	47:32	分配给对端 VL2 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x61	0x0
Cfg Remote crd vl1	31:16	分配给对端 VL1 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x60	0x0
Cfg Remote crd vl0	15:0	分配给对端 VL0 的初始信用证数量 (单位: CELL)	RW_DE0_EO	0x60	0x0

### 3. Data Link Retry Cfg

表 D-113 Data Link Retry Cfg 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Wait Retry Ack Timeout_L	31:0	重传时等待重传应答的最大时间 (单位: us)	RW_DE0_EO	0x68	0x40

### 4. Data Link Performance Optimization Cfg

表 D-114 Data Link Performance Optimization Cfg 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Cfg Ack Llcrd	31:16	强制添加 Crd_Ack Block 返回 ACK 的水线配置 (超过此水线会反压业务包并生成 Crd_Ack Block)	RW_DE0_EO	0x70	--

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Cfg Ack Return	15:0	通过 Crd_Ack Block 返回 ACK 的水线配置（超过此水线才会生成 Crd_Ack Block）	RW_DE0_EO	0x70	--
Cfg Crd Vn Llcrd	255:0	强制添加 Crd_Ack Block 返回信用证的水线配置（超过此水线会反压业务包并生成 Crd_Ack Block），16 通道，每通道 16bit，各个 VL 的水线配置值分别为：{Waterlevel_VL15, Waterlevel_VL14, ..., Waterlevel_VL0}	RW_DE0_EO	0x71	--
Cfg Crd Vn Return	255:0	通过 Crd_Ack Block 返回信用证的水线配置（超过此水线才会生成 Crd_Ack Block），16 通道，每通道 16bit，各个 VL 的水线配置值分别为：{Waterlevel_VL15, Waterlevel_VL14, ..., Waterlevel_VL0}	RW_DE0_EO	0x79	--

## 5. Data Link MarkErr Cfg

表 D-115 Data Link MarkErr Cfg 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:1		RO_DE0_EO	0x81	NA
Cfg MarkErr En	0	业务包标错启用	RW_DE0_EO	0x81	0x0

## 6. Data Link Control Reserve 1

表 D-116 Data Link Control Reserve 1 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:0	Reserved	RO_DE0_EO	0x82	0x0

## 7. PHY Link Control 1

Start Offset Address:0x90

表 D-117 PHY Link Control 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Target Link Speed	3:0	端口目标速率。 4'h0: Data Rate0 4'h1: Data Rate1 4'h2: Data Rate2 4'h3: Data Rate3 4'h4: Data Rate4 4'h5: Data Rate5 4'h6: Data Rate6 4'h7: Data Rate7 4'h8: Data Rate8 4'h9: Data Rate9 Others: Reserved.	RW_DE0_EO
Reserved	7:4	Reserved	RO_DE0_EO
Target TX Link Width	13:8	目标 TX 链路宽度。  注: 在 LMSM 启动之前, 目标 TX 链路宽度必须等于目标 RX 链路宽度。 6'h0: Reserved 6'h1: x1 6'h2: x2 6'h4: x4 6'h8: x8 Others: Reserved  在 LMSM 训练到期望的速率和链路宽度后, 软件可以设置目标 TX 链路宽度或者目标 RX 链路宽度, 然后启动 LMSM 进入 Retrain 状态来改变 TX 或者 RX 链路宽度。	RW_DE0_EO
Reserved	15:14	Reserved	RO_DE0_EO
Target RX Link Width	21:16	目标 RX 链路宽度。  注: 在 LMSM 启动之前, 目标 TX 链路宽度必须等于目标 RX 链路宽度。 6'h0: Reserved 6'h1: x1 6'h2: x2 6'h4: x4 6'h8: x8 Others: Reserved.	RW_DE0_EO
Change Link width not Retrain Link	22	向此位写入 1 以触发不进入 Retrain 状态来改变链路宽度。  读取此位总是返回 0。	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Reserved	23	Reserved	RO_DE0_EO
PHY Mode Control	27:24	物理层工作模式配置。 4'h0: PHY-Mode1 4'h1: PHY-Mode2 Others: Reserved	RW_DE0_EO
Reserved	28	Reserved	RO_DE0_EO
Reserved	30:29	Reserved	RO_DE0_EO
Retrain Link	31	写 1 会使得 LMSM 进入 Retrain 状态。 读取此位总是返回 0。 注：如果 LMSM 当前已经处于 Retrain 状态或者 Config 状态，允许重新进入 Retrain 状态，但不强制要求。	RW_DE0_EO

## 8. PHY Link Control 2

Start Offset Address:0x91

表 D-118 PHY Link Control 2 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	3:0	Reserved	RO_DE0_EO
EQ Mode control	5:4	物理层的 EQ 模式设置。 2'b00: Full_EQ 2'b01: Only_Highest_Data_Rate_EQ 2'b10: Skip_EQ Others: Reserved  设备级复位或 Port 级复位后，此位的默认值为 1。	RW_DE0_EO
De-emphasis value	6	当链路工作在 4.0 Gbps 速率时，此比特控制链路 De-emphasis 值。 1'b0: -3.5 dB 1'b1: -6 dB  对别的速率此位无效。	RW_DE0_EO
Perform Equalization	7	当链路工作在 Data Rate1 及以上速率时，此比特写 1 启动均衡过程。	RW_DE0_EO
Reserved	15:8	Reserved	RO_DE0_EO
Over fibre mode Enable	16	启用光链路模式。 1'b0: 不启用光链路模式	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		1'b1: 启用光链路模式 1'b0: 不启用光链路模式	
Fix Data Rate mode Enable	17	启用固定速率模式。 1'b0: 不启用固定速率模式 1'b1: 启用固定速率模式 当启用固定速率模式时，固定速率由 PHY Link Control 1 寄存器中的 Target Link Speed 域段配置。	RW_DE0_EO
Bypass Probe Enable	18	启用旁路 Probe 模式。 1'b0: Enable Probe 状态 1'b1: Bypass Probe 状态	RW_DE0_EO
Port Type Nego Enable	19	启用端口类型协商。 1'b0: Disable Port type negotiation 1'b1: Enable Port type negotiation	RW_DE0_EO
Reserved	20	Reserved	RO_DE0_EO
Link Disable	21	启用 Link Disable 功能 1'b0: Not request enter Link.Disable state 1'b1: Request enter Link.Disable state	RW_DE0_EO
Reserved	31:22	Reserved	RO_DE0_EO

## 9. PHY Link Control 3

Start Offset Address:0x92

表 D-119 PHY Link Control 3 寄存器

Bit Location	Register Description	Attribute
31:0	Reserved	RO_DE0_EO

## 10. PHY Link Control 4

Start Offset Address:0x93

表 D-120 PHY Link Control 4 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	15:0	Reserved	RO_DE0_EO
Reserved	31:16	Reserved	RO_DE0_EO

## 11. PHY Link Control 5

Start Offset Address:0x94

表 D-121 PHY Link Control 5 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	15:0	Reserved	RO_DE0_EO
Precode Enable Vector	31:16	指示启用对应端口的预编码 每一位 bit 值为 1'b1 表示对应的链路速度启用预编码，否则不启用预编码。 Bit16: Data Rate0 Bit17: Data Rate1 ... Bit25: Data Rate9 Bit26-Bit31: Reserved 该域段的默认值为 16'hFFF0	RW_DE0_EO

## 12. PHY Link Control 6

Start Offset Address:0x95

表 D-122 PHY Link Control 6 寄存器

Field Name	Bit Location	Register Description	Attribute
FEC Mode for Data Rate0	3:0	设置数据速率 0 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该域段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate1	7:4	设置数据速率 1 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该域段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate2	11:8	设置数据速率 2 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该域段的默认值为 4'h0	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
FEC Mode for Data Rate3	15:12	设置数据速率 3 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate4	19:16	设置数据速率 4 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate5	23:20	设置数据速率 5 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate6	27:24	设置数据速率 6 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO
FEC Mode for Data Rate7	31:28	设置数据速率 7 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO

## 13. PHY Link Control 7

Start Offset Address:0x96

表 D-123 PHY Link Control 7 寄存器

Field Name	Bit Location	Register Description	Attribute
FEC Mode for Data Rate8	3:0	设置数据速率 8 的 FEC 模式	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	
FEC Mode for Data Rate9	7:4	设置数据速率 9 的 FEC 模式 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved 该字段的默认值为 4'h0	RW_DE0_EO
Reserved	31:8	Reserved	RO_DE0_EO

## 14. PHY Link Control 8

Start Offset Address:0x97

表 D-124 PHY Link Control 8 寄存器

Field Name	Bit Location	Register Description	Attribute
BCRC Mode for Data Rate0	3:0	设置数据速率 0 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate1	7:4	设置数据速率 1 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate2	11:8	设置数据速率 2 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate3	15:12	设置数据速率 3 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		Others: Reserved 该字段的默认值为 4'h1	
BCRC Mode for Data Rate4	19:16	设置数据速率 4 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate5	23:20	设置数据速率 5 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate6	27:24	设置数据速率 6 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate7	31:28	设置数据速率 7 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO

## 15. PHY Link Control 9

Start Offset Address:0x98

表 D-125 PHY Link Control 9 寄存器

Field Name	Bit Location	Register Description	Attribute
BCRC Mode for Data Rate8	3:0	设置数据速率 8 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved 该字段的默认值为 4'h1	RW_DE0_EO
BCRC Mode for Data Rate9	7:4	设置数据速率 9 的 BCRC 模式 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		该字段的默认值为 4'h1	
Reserved	31:8	Reserved	RO_DE0_EO

## 16. PHY Link Control 10

Start Offset Address:0x99

表 D-126 PHY Link Control 10 寄存器

Field Name	Bit Location	Register Description	Attribute
FEC Interleave Enable Vector	15:0	设置对应的速率是否启用 FEC 交织。 每一位 bit 值为 1'b1 时表示对应的速率启用 FEC 交织，否则不启用 FEC 交织。 Bit0: Data Rate0 Bit1: Data Rate1 ... Bit9: Data Rate9 Bit10-Bit15: Reserved 缺省值由具体实现决定	RW_DE0_EO
Reserved	31:16	Reserved	RO_DE0_EO

## 17. PHY Link Control 11

Start Offset Address:0x9a

表 D-127 PHY Link Control 11 寄存器

Field Name	Bit Location	Register Description	Attribute
Skip EQ Coarsetune Enable	15:0	设置对应的速率是否使能 Skip EQ Coarsetune。 每一位 bit 值为 1'b1 时表示对应的速率使能 Skip EQ Coarsetune，否则不使能 Skip EQ Coarsetune。 Bit0: 0 Bit1: Data Rate1 ... Bit9: Data Rate9 Bit10-Bit15: Reserved 缺省值由具体实现决定	RW_DE0_EO
Reserved	31:16	Reserved	RO_DE0_EO

### D.6.2.3 Link Status

#### 1. Data Link Status

表 D-128 Data Link Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:1	Reserved	RO_DE0_EO	0xa0	NA
RXBUF_VL_SH ARE_pro_sta	0	本端 RXBUF 是否支持 VL 共享  1'b1: 支持; 1'b0: 不支持	RO_DE0_EO	0xa0	NA
Reserved	31:16	Reserved	RO_DE0_EO	0xa1	NA
CTRL_ACK_GR AIN_SIZE_pro_sta	15:8	Crd_Ack Block 返回 ACK 支持的粒度：数据包头返还信用证支持的粒度：每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa1	NA
DATA_ACK_GR AIN_SIZE_pro_sta	7:0	业务包头返还 ACK 的粒度：数据包头返还信用证支持的粒度：每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa1	NA
CTRL_CREDIT_GRAIN_SIZE_pro_0_sta	31:0	Crd_Ack Block 支持 VL0~VL3 的 credit 返还粒度（单位：CELL）  每个 byte 对应一个 VL 的 credit 返还粒度{VL3,VL2,VL1,VL0}  每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa2	NA
CTRL_CREDIT_GRAIN_SIZE_pro_1_sta	31:0	Crd_Ack Block 支持 VL4~VL7 的 credit 返还粒度（单位：CELL）  每个 byte 对应一个 VL 的 credit 返还粒度{VL7,VL6,VL5,VL4}  每 bit 对应一个 CELL 数量，分别为{128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa3	NA

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
CTRL_CREDIT_GRAIN_SIZE_pro_2_sta	31:0	Crd_Ack Block 支持 VL8~VL11 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL11,VL10, VL9,VL8} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa4	NA
CTRL_CREDIT_GRAIN_SIZE_pro_3_sta	31:0	Crd_Ack Block 支持 VL12~VL15 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL15,VL14, VL13,VL12} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa5	NA
DATA_CREDIT_GRAIN_SIZE_pro_0_sta	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL0~VL3 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL3,VL2,VL1,VL0} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa6	NA
DATA_CREDIT_GRAIN_SIZE_pro_1_sta	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL4~VL7 的 credit 返还粒度 (单位: CELL) 每个 byte 对应一个 VL 的 credit 返还粒度 {VL7,VL6,VL5,VL4} 每 bit 对应一个 CELL 数量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa7	NA
DATA_CREDIT_GRAIN_SIZE_	31:0	业务包 Link Packet Header/Link Block Header	RO_DE0_EO	0xa8	NA

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
pro_2_sta		中支持 VL8~VL11 的 credit 返还粒度 (单位: CELL )  每个 byte 对应一个 VL 的 credit 返还粒度 {VL11,VL10,VL9,VL8}  每 bit 对应一个 CELL 数 量, 分别为 {128,64,32,16,8,4,2,1}			
DATA_CREDIT_GRAIN_SIZE_pro_3_sta	31:0	业务包 Link Packet Header/Link Block Header 中支持 VL12~VL15 的 credit 返还粒度 (单位: CELL )  每个 byte 对应一个 VL 的 credit 返还粒度 {VL15,VL14,VL13,VL12}  每 bit 对应一个 CELL 数 量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xa9	NA
PACKET_MIN_INTERVAL_sta	31:24	支持最小包间隔, 单 位:FLIT	RO_DE0_EO	0xaa	NA
FLOW_CTRL_SIZE_pro_sta	23:16	支持信用流控最小单元 CELL 对应的 FLIT 数量: 每 bit 对应一个 FLIT 数 量, 分别为 {128,64,32,16,8,4,2,1}	RO_DE0_EO	0xaa	NA
VL_ENABLE_pro_sta	15:0	是否支持虚通道	RO_DE0_EO	0xaa	NA
Reserved	31:20	Reserved	RO_DE0_EO	0xab	NA
RETRY_BUF_DEPTH_sta	19:0	重传 BUF 深度	RO_DE0_EO	0xab	NA

## 2. Data Link State Machine Status

表 D-129 Data Link State Machine Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Vaule
Reserved	31:12	Reserved	RO_DE0_EO	0xac	NA

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Retry Ack Status	11:8	重传应答状态机状态 4'd0: NORMAL 状态 4'd1: ACK 状态 Others: Undefined	RO_DE0_EO	0xac	NA
Retry Req Status	7:4	重传请求状态机状态 4'd0: NORMAL 状态 4'd1: REQ 状态 4'd2: WAIT 状态 4'd3: RETRAIN 状态 4'd4: ERROR 状态 Others: Undefined	RO_DE0_EO	0xac	NA
Link Status	3:0	状态管理状态机状态 4'd0: DLL_Disabled 状态 4'd1: DLL_Param_Init 状态 4'd2: DLL_Credit_Init 状态 4'd3: DLL_Normal 状态 Others: Undefined	RO_DE0_EO	0xac	NA

## 3. Data Link status Reserve 1

表 D-130 Data Link status Reserve 1 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:0	Reserved	RO_DE0_EO	0xad	0x0

## 4. Data Link status Reserve 2

表 D-131 Data Link status Reserve 2 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Reserved	31:0	Reserved	RO_DE0_EO	0xae	0x0

## 5. Data Link status Reserve 3

**表 D-132 Data Link status Reserve 3 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Vaule</b>
Reserved	31:0	Reserved	RO_DE0_EO	0xaf	0x0

## 6. Data Link status Reserve 4

**表 D-133 Data Link status Reserve 4 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Vaule</b>
Reserved	31:0	Reserved	RO_DE0_EO	0xb0	0x0

## 7. Port Link Status 0

**表 D-134 Port Link Status 0 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Vaule</b>
PHY Link state	0	LMSM 是否处于 LinkUp 状态。 1'b1: LinkUp 1'b0: LinkDown	RO_DE0_EO	0xc0	
Data Link Layer Link State	1	指示数据链路状态机状态。 1'b1: DLL_Normal 1'b0: Others	RO_DE0_EO	0xc0	
Reserved	31:2	Reserved	RO_DE0_EO	0xc0	

## 8. PHY Link Status 1

Start Offset Address:0xc1

**表 D-135 PHY Link Status 1 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Current Link Speed	3:0	指示端口当前的链路速率。 4'h0: Data Rate0 4'h1: Data Rate1 4'h2: Data Rate2 4'h3: Data Rate3 4'h4: Data Rate4 4'h5: Data Rate5 4'h6: Data Rate6 4'h7: Data Rate7	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		4'h8: Data Rate8 4'h9: Data Rate9 Others: Reserved	
Current TX Link Width	8:4	指示当前 TX 链路宽度 ( xN-对应 N 个 Lanes ) 。 5'h0: Reserved 5'h1: x1 5'h2: x2 5'h4: x4 5'h8: x8 Others: Reserved	RO_DE0_EO
Reserved	11:9	Reserved	RO_DE0_EO
Current RX Link Width	16:12	指示当前 RX 链路宽度 ( xN-对应 N 个 Lanes ) 。 5'h0: Reserved 5'h1: x1 5'h2: x2 5'h4: x4 5'h8: x8 Others: Reserved	RO_DE0_EO
Reserved	19:17	Reserved	RO_DE0_EO
Current LMSM State	25:20	指示当前 LMSM 状态。	RO_DE0_EO
Reserved	27:26	Reserved	RO_DE0_EO
Current Port Type	28	指示当前端口类型。 1'b0: Secondary Port 1'b1: Primary Port	RO_DE0_EO
Reserved	31:29	Reserved	RO_DE0_EO

## 9. PHY Link Status 2

Start Offset Address:0xc2

表 D-136 PHY Link Status 2 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	1:0	Reserved	RO_DE0_EO
Current Precode State	3:2	指示当前数据速率的预编码状态。每 bit 值为 1'b1 表示对应的 Precode 处于开启状态。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Current FEC Mode	7:4	指示当前数据速率的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	RO_DE0_EO
Current EQ Mode	11:8	指示当前 EQ 模式。 2'h0: Full_EQ 2'h1: Only_Highest_Data_Rate_EQ 2'h2: Skip_EQ Others: Reserved	RO_DE0_EO
Current BCRC Mode	15:12	指示当前速率的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	16	指示当前速率的 FEC 交织状态。 值为 1'b1 表示 FEC 交织打开。	RO_DE0_EO
Reserved	31:17	Reserved	RO_DE0_EO

## 10. PHY Link Status 3

Start Offset Address:0xc3

表 D-137 PHY Link Status 3 寄存器

Field Name	Bit Location	Register Description	Attribute
PHY Link Status 3	31:0	链路离开 Link_Active 状态时记录的信息	RW1C_DE0_EO

## 11. PHY Link Status 4

Start Offset Address:0xc4

表 D-138 PHY Link Status4 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

## 12. Data Link Error status

表 D-139 Data Link Error Status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	31:0	Reserved	RO_DE0_EO	0xc5	NA

## 13. PHY Link Error status

表 D-140 PHY Link Error status 寄存器

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Deskew_buf_overflow	0	Deskew 缓冲区溢出	RO_DE0_EO	0xc6	0x0
eBCH decode error	1	eBCH decode 错误	RO_DE0_EO	0xc6	0x0
Reserved	31:2	Reserved	RO_DE0_EO	0xc6	0x0

**D.6.3 PORT\_CAP2\_LINK\_LOG****D.6.3.1 Slice Header**

表 D-141 LINK\_LOG 寄存器

Field Name	Value
Slice Version	0x0
Slice Used Size	0x11

**D.6.3.2 Flit Error Log 1**

Start Offset Address:0x1

表 D-142 Flit Error Log 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Flit Error Log Valid	0	指示 Flit 错误日志是否有效。 当错误记录在此寄存器中时，此位值为 1。 向此位写入 1 将清除此位，或者如果更多 Flit Error Log 寄存器组的条目有效（如 bit 13）被设置，则将 Flit Error Log 1 Register 和 Flit Error Log 2 Register 加载为下一个最旧的日志条目。 缺省值为 0。	RO_DE0_EO
Reserved	31:1	Reserved	RO_DE0_EO

**D.6.3.3 Flit Error Log 2**

Start Offset Address:0x2

表 D-143 Flit Error Log 2 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.3.4 Flit Error Counter Control**

Start Offset Address:0x3

表 D-144 Flit Error Counter Control 寄存器

Field Name	Bit Location	Register Description	Attribute
Flit Error Counter Enable	0	启用 Flit 错误计数器。 设置此位为 1 时，启用链路中的 Flit 错误计数器。 此位清 0 将停止 Flit 错误计数器。 缺省值为 0。	RW_DE0_EO
Reserved	31:1	Reserved	RO_DE0_EO

**D.6.3.5 Flit Error Counter Status**

Start Offset Address:0x4

表 D-145 Flit Error Counter Status 寄存器

Field Name	Bit Location	Register Description	Attribute
Flit Error Counter	31:0	Flit 错误计数器	RW1C_DE0_EO

**D.6.3.6 LTB Error Log 1**

Start Offset Address:0x5

表 D-146 LTB Error Log1 寄存器

Field Name	Bit Location	Register Description	Attribute
LTB Error Log1	31:0	DLTB Error count	RW1C_DE0_EO

**D.6.3.7 LTB Error Log 2**

Start Offset Address:0x6

**表 D-147 LTB Error Log2 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
LTB Error Log2	31:0	CLTB Error count	RW1C_DE0_EO

**D.6.3.8 LTB Error Log 3**

Start Offset Address:0x7

**表 D-148 LTB Error Log 3 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
LTB Error Log3	31:0	ELTB Error count	RW1C_DE0_EO

**D.6.3.9 LTB Error Log 4**

Start Offset Address:0x8

**表 D-149 LTB Error Log 4 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
LTB Error Log4	31:0	RLTB Error count	RW1C_DE0_EO

**D.6.3.10 LTB Error Log 5**

Start Offset Address:0x9

**表 D-150 LTB Error Log 5 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Reserved	31:0	Reserved	-

**D.6.3.11 LTB Error Counter Control**

Start Offset Address:0xa

**表 D-151 LTB Error Counter Control 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
LTB Error Counter Control	31:6	Reserved	RO_DE0_EO
	5	LTB Error Log 5 Count enable	RW_DE0_EO
	4	LTB Error Log 4 Count enable	RW_DE0_EO
	3	LTB Error Log 3 Count enable	RW_DE0_EO
	2	LTB Error Log 2 Count enable	RW_DE0_EO
	1	LTB Error Log 1 Count enable	RW_DE0_EO

Field Name	Bit Location	Register Description	Attribute
	0	LTB Error Log 0 Count enable	RW_DE0_EO

### D.6.3.12 LTB Error Counter Status

Start Offset Address:0xb

表 D-152 LTB Error Counter Status 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.3.13 BER Measurement Control

Start Offset Address:0xc

表 D-153 BER Measurement Control 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.3.14 BER Measurement Status 1

Start Offset Address:0xd

表 D-154 BER Measurement Status 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.3.15 BER Measurement Status 2

Start Offset Address: 0xe

表 D-155 BER Measurement Status 2 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.3.16 BER Measurement Status 3

Start Offset Address:0xf

表 D-156 BER Measurement Status 3 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.3.17 BER Measurement Status 4**

Start Offset Address:0x10

**表 D-157 BER Measurement Status 4 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.4 PORT\_CAP3\_RSVD**

Reserved

**D.6.5 PORT\_CAP4\_DATA\_RATE1****表 D-158 DATA RATE1 Slice Header**

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

**D.6.5.1 Control 1**

Start Offset Address:0x1

**表 D-159 DATA RATE1 Control 1 寄存器**

Field Name	Bits	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.5.2 Control 2**

Start Offset Address:0x2

**表 D-160 DATA\_RATE1 Control 2 寄存器**

<b>Field Name</b>	<b>Bits</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.5.3 Control 3**

Start Offset Address:0x3

Reserved

**D.6.5.4 Control 4**

Start Offset Address:0x4

Reserved

**D.6.5.5 Control 5**

Start Offset Address:0x5

Reserved

**D.6.5.6 State 1**

Start Offset Address:0x6

**表 D-161 DATA\_RATE1 State 1 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Equalization Coarsestune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsestune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 1 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Reserved	31:19	Reserved	RO_DE0_EO

### D.6.5.7 State 2

Start Offset Address:0x7

表 D-162 DATA\_RATE1 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

### D.6.5.8 State 3

Start Offset Address:0x8

Reserved

### D.6.5.9 State 4

Start Offset Address:0x9

Reserved

## D.6.6 PORT\_CAP5\_DATA\_RATE2

### D.6.6.1 Slice Header

表 D-163 DATA\_RATE2 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

**D.6.6.2 Control 1**

Start Offset Address:0x1

**表 D-164 DATA\_RATE2 Control 1 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.6.3 Control 2**

Start Offset Address:0x2

**表 D-165 DATA\_RATE2 Control 2 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.6.4 Control 3**

Start Offset Address:0x3

Reserved

**D.6.6.5 Control 4**

Start Offset Address:0x4

Reserved

**D.6.6.6 Control 5**

Start Offset Address:0x5

Reserved

**D.6.6.7 State 1**

Start Offset Address:0x6

**表 D-166 DATA\_RATE2 State 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 2 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.6.8 State 2

Start Offset Address:0x7

表 D-167 DATA\_RATE2 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

**D.6.6.9 State 3**

Start Offset Address:0x8

Reserved

**D.6.6.10 State 4**

Start Offset Address:0x9

**表 D-168 DATA\_RATE2 State 4 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.7 PORT\_CAP6\_DATA\_RATE3****D.6.7.1 Slice Header****表 D-169 DATA\_RATE3 Slice Header**

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

**D.6.7.2 Control 1**

Start Offset Address:0x1

**表 D-170 DATA\_RATE3 Control 1 寄存器**

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.7.3 Control 2**

Start Offset Address:0x2

**表 D-171 DATA\_RATE3 Control 2 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.7.4 Control 3**

Start Offset Address:0x3

Reserved

**D.6.7.5 Control 4**

Start Offset Address:0x4

Reserved

**D.6.7.6 Control 5**

Start Offset Address:0x5

Reserved

**D.6.7.7 State 1**

Start Offset Address:0x6

**表 D-172 DATA\_RATE3 State 1 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Equalization Coarse tune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 3 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Reserved	31:19	Reserved	RO_DE0_EO

### D.6.7.8 State 2

Start Offset Address:0x7

表 D-173 DATA\_RATE3 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

### D.6.7.9 State 3

Start Offset Address:0x8

Reserved

### D.6.7.10 State 4

Start Offset Address:0x9

Reserved

## D.6.8 PORT\_CAP7\_DATA\_RATE4

### D.6.8.1 Slice Header

表 D-174 DATA\_RATE4 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

**D.6.8.2 Control 1**

Start Offset Address:0x1

**表 D-175 DATA\_RATE4 Control 1 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.8.3 Control 2**

Start Offset Address:0x2

**表 D-176 DATA\_RATE4 Control 2 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.8.4 Control 3**

Start Offset Address:0x3

Reserved

**D.6.8.5 Control 4**

Start Offset Address:0x4

Reserved

**D.6.8.6 Control 5**

Start Offset Address:0x5

Reserved

**D.6.8.7 State 1**

Start Offset Address:0x6

**表 D-177 DATA\_RATE4 State 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsetune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsetune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 4 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.8.8 State 2

Start Offset Address:0x7

表 D-178 DATA\_RATE4 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

#### D.6.8.9 State 3

Start Offset Address:0x8

Reserved

**D.6.8.10 State 4**

Start Offset Address:0x9

Reserved

**D.6.9 PORT\_CAP8\_DATA\_RATE5****D.6.9.1 Slice Header****表 D-179 DATA\_RATE5 Slice Header**

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

**D.6.9.2 Control 1**

Start Offset Address:0x1

**表 D-180 DATA\_RATE5 Control 1 寄存器**

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.9.3 Control 2**

Start Offset Address:0x2

**表 D-181 DATA\_RATE5 Control 2 寄存器**

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO

Field Name	Bit	Register Description	Attribute
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

#### D.6.9.4 Control 3

Start Offset Address:0x3

Reserved

#### D.6.9.5 Control 4

Start Offset Address:0x4

Reserved

#### D.6.9.6 Control 5

Start Offset Address:0x5

Reserved

#### D.6.9.7 State 1

Start Offset Address:0x6

表 D-182 DATA\_RATE5 State 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Equalization Coarsetune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 5 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

**D.6.9.8 State 2**

Start Offset Address:0x7

**表 D-183 DATA\_RATE5 State 2 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

**D.6.9.9 State 3**

Start Offset Address:0x8

Reserved

**D.6.9.10 State 4**

Start Offset Address:0x9

Reserved

**D.6.10 PORT\_CAP9\_DATA\_RATE6****D.6.10.1 Slice Header****表 D-184 DATA\_RATE6 Slice Header**

<b>Field Name</b>	<b>Value</b>
Slice Version	0x0
Slice Used Size	0xA

**D.6.10.2 Control 1**

Start Offset Address:0x1

**表 D-185 DATA\_RATE6 Control 1 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.10.3 Control 2**

Start Offset Address:0x2

**表 D-186 DATA\_RATE6 Control 2 寄存器**

<b>Field Name</b>	<b>Bit</b>	<b>Register Description</b>	<b>Attribute</b>
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

**D.6.10.4 Control 3**

Start Offset Address:0x3

Reserved

**D.6.10.5 Control 4**

Start Offset Address:0x4

Reserved

**D.6.10.6 Control 5**

Start Offset Address:0x5

Reserved

**D.6.10.7 State 1**

Start Offset Address:0x6

**表 D-187 DATA\_RATE6 State 1 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 6 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.10.8 State 2

Start Offset Address:0x7

表 D-188 DATA\_RATE6 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

#### D.6.10.9 State 3

Start Offset Address:0x8

Reserved

#### D.6.10.10 State 4

Start Offset Address:0x9

Reserved

## D.6.11 PORT\_CAP10\_DATA\_RATE7

### D.6.11.1 Slice Header

表 D-189 DATA\_RATE7 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

### D.6.11.2 Control 1

Start Offset Address:0x1

表 D-190 DATA\_RATE7 Control 1 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

### D.6.11.3 Control 2

Start Offset Address:0x2

表 D-191 DATA\_RATE7 Control 2 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

#### D.6.11.4 Control 3

Start Offset Address:0x3

Reserved

#### D.6.11.5 Control 4

Start Offset Address:0x4

Reserved

#### D.6.11.6 Control 5

Start Offset Address:0x5

Reserved

#### D.6.11.7 State 1

Start Offset Address:0x6

表 D-192 DATA\_RATE7 State 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsetune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsetune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的 主动阶段已成功完成。	ROS_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		主动阶段已成功完成。 缺省值为 0。	
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 7 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120,T=2) 4'h2: RS (128, 120,T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.11.8 State 2

Start Offset Address:0x7

表 D-193 DATA\_RATE7 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

### D.6.11.9 State 3

Start Offset Address:0x8

Reserved

### D.6.11.10 State 4

Start Offset Address:0x9

Reserved

## D.6.12 PORT\_CAP11\_DATA\_RATE8

### D.6.12.1 Slice Header

表 D-194 DATA\_RATE8 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

### D.6.12.2 Control 1

Start Offset Address:0x1

表 D-195 DATA\_RATE8 Control 1 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO

Field Name	Bit	Register Description	Attribute
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

#### D.6.12.3 Control 2

Start Offset Address:0x2

表 D-196 DATA\_RATE8 Control 2 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

#### D.6.12.4 Control 3

Start Offset Address:0x3

Reserved

#### D.6.12.5 Control 4

Start Offset Address:0x4

Reserved

#### D.6.12.6 Control 5

Start Offset Address:0x5

Reserved

**D.6.12.7 State 1**

Start Offset Address:0x6

**表 D-197 DATA\_RATE8 State 1 寄存器**

<b>Field Name</b>	<b>Bit Location</b>	<b>Register Description</b>	<b>Attribute</b>
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsestune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarsestune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 8 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved	RO_DE0_EO

Field Name	Bit Location	Register Description	Attribute
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.12.8 State 2

Start Offset Address:0x7

表 D-198 DATA\_RATE8 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

#### D.6.12.9 State 3

Start Offset Address:0x8

Reserved

#### D.6.12.10 State 4

Start Offset Address:0x9

Reserved

## D.6.13 PORT\_CAP12\_DATA\_RATE9

### D.6.13.1 Slice Header

表 D-199 DATA\_RATE9 Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0xA

### D.6.13.2 Control 1

Start Offset Address:0x1

表 D-200 DATA\_RATE9 Control 1 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane0	3:0	本地端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane0	7:4	远端端口 Lane0 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane1	11:8	本地端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane1	15:12	远端端口 Lane1 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane2	19:16	本地端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane2	23:20	远端端口 Lane2 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane3	27:24	本地端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane3	31:28	远端端口 Lane3 的初始 TX 均衡预设值。	RW_DE0_EO

### D.6.13.3 Control 2

Start Offset Address:0x2

表 D-201 DATA\_RATE9 Control 2 寄存器

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane4	3:0	本地端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane4	7:4	远端端口 Lane4 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane5	11:8	本地端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane5	15:12	远端端口 Lane5 的初始 TX 均衡预设值。	RW_DE0_EO
Local_TX_Preset_Lane6	19:16	本地端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane6	23:20	远端端口 Lane6 的初始 TX 均衡预设值。	RW_DE0_EO

Field Name	Bit	Register Description	Attribute
Local_TX_Preset_Lane7	27:24	本地端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO
Remote_TX_Preset_Lane7	31:28	远端端口 Lane7 的初始 TX 均衡预设值。	RW_DE0_EO

#### D.6.13.4 Control 3

Start Offset Address:0x3

Reserved

#### D.6.13.5 Control 4

Start Offset Address:0x4

Reserved

#### D.6.13.6 Control 5

Start Offset Address:0x5

Reserved

#### D.6.13.7 State 1

Start Offset Address:0x6

表 D-202 DATA\_RATE9 State 1 寄存器

Field Name	Bit Location	Register Description	Attribute
Equalization Complete	0	值为 1 时表示该速率下 TX 均衡过程已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune phase Complete	1	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已完成。 缺省值为 0。	ROS_DE0_EO
Equalization Coarse tune Phase Successful	2	值为 1 时表示该速率下 TX 均衡过程的粗调阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Passive Phase Successful	3	值为 1 时表示该速率下 TX 均衡过程的被动阶段已成功完成。 缺省值为 0。	ROS_DE0_EO
Equalization Active Phase Successful	4	值为 1 时表示该速率下 TX 均衡过程的主动阶段已成功完成。	ROS_DE0_EO

Field Name	Bit Location	Register Description	Attribute
		缺省值为 0。	
Link Equalization Request	5	硬件设置此比特为 1 以请求执行针对 Data Rate 9 的链路均衡过程。 缺省值为 0。	RW1CS_DE0_EO
Transmitter Precoding On Request	6	指示远端端口是否要求本地端口为此速率启用 TX 预编码。 缺省值为 0	RO_DE0_EO
Reserved	7	Reserved	RO_DE0_EO
Current precode state	9:8	指示此速率的预编码状态。 值为 1 时代表开启了 Precode。 Bit0: TX Precode Bit1: RX Precode	RO_DE0_EO
Current FEC Mode	13:10	指示此速率下的 FEC 模式。 4'h0: FEC Bypass 4'h1: RS (128, 120, T=2) 4'h2: RS (128, 120, T=4) Others: Reserved	RO_DE0_EO
Current BCRC Mode	17:14	指示此速率下的 BCRC 模式。 4'h0: No BCRC 4'h1: CRC30 Others: Reserved	RO_DE0_EO
Current FEC Interleave state	18	指示此速率下的 FEC 交织状态。 值为 1 时代表开启了 FEC 交织。	RO_DE0_EO
Reserved	31:19	Reserved	RO_DE0_EO

#### D.6.13.8 State 2

Start Offset Address:0x7

表 D-203 DATA\_RATE9 State 2 寄存器

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane0	3:0	本地端口 Lane0 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane1	7:4	本地端口 Lane1 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane2	11:8	本地端口 Lane2 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane3	15:12	本地端口 Lane3 当前的均衡预设值。	RO_DE0_EO

Field Name	Bit	Register Description	Attribute
Current_Local_TX_Preset_Lane4	19:16	本地端口 Lane4 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane5	23:20	本地端口 Lane5 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane6	27:24	本地端口 Lane6 当前的均衡预设值。	RO_DE0_EO
Current_Local_TX_Preset_Lane7	31:28	本地端口 Lane7 当前的均衡预设值。	RO_DE0_EO

### D.6.13.9 State 3

Start Offset Address:0x8

Reserved

### D.6.13.10 State 4

Start Offset Address:0x9

Reserved

## D.6.14 PORT\_CAP13\_RSVD

Reserved

## D.6.15 PORT\_CAP14\_EYE\_MONITOR

### D.6.15.1 Slice Header

表 D-204 EYE MONITOR Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x4+N (N=Max Lane Number)

### D.6.15.2 Capability

Start Offset Address:0x1

表 D-205 EYE MONITOR Capability 寄存器

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.15.3 Control 1

Start Offset Address:0x2

**表 D-206 EYE MONITOR Control 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
Eye Monitor Test	7:0	眼图测试。 Bitmap 对应每一个 Lane ( Bit0 对应 Lane0 )。 软件对该寄存器相应比特写 1 来启动对应 Lane 的眼图测试。	RW_DE0_EO
Reserved	31:8	Reserved	RO_DE0_EO

**D.6.15.4 Control 2**

Start Offset Address:0x3

**表 D-207 EYE MONITOR Control 2 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.15.5 LaneN State**

Start Offset Address:0x4+Lane number (0&lt;=Lane number&lt;8)

**表 D-208 EYE MONITOR Control 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
Eye Monitor Test Execution Status	2:0	眼图测试执行状态。 软件可以读取此寄存器以检查当前眼图测试的执行状态，定义如下： 3'b000: Idle 3'b001: Abort 3'b010: In progress 3'b011: Timeout 3'b100: Successful Done	RO_DE0_EO
Reserved	15:3	Reserved	RO_DE0_EO
Eye Monitor Test Result	31:16	Eye Monitor test result。	RO_DE0_EO

## D.6.16 PORT\_CAP15\_QDLWS

### D.6.16.1 Slice Header

表 D-209 QDLWS Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	0x4

### D.6.16.2 Capability

Start Offset Address:0x1

表 D-210 QDLWS Capability 寄存器

Field Name	Bit Location	Register Description	Attribute
Asymmetry Link Width Change Support	0	不对称链路宽度改变支持 0: 不支持 1: 支持	RO_DE0_EO
Reserved	31:1	Reserved	RO_DE0_EO

### D.6.16.3 Control

Start Offset Address:0x2

表 D-211 QDLWS Control 寄存器

Field Name	Bit Location	Register Description	Attribute
Global Enable for QDLWS	0	QDLWS 启用控制 0: 不启用 1: 启用	RW_DE0_EO
TX QDLWS Enable	1	TX QDLWS 启用控制 0: 不启用 1: 启用	RW_DE0_EO
RX QDLWS Enable	2	RX QDLWS 启用控制 0: 不启用 1: 启用	RW_DE0_EO
Reserved	31:3	Reserved	RO_DE0_EO

**D.6.16.4 State**

Start Offset Address:0x3

**表 D-212 QDLWS State 寄存器**

Field Name	Bit Location	Register Description	Attribute
QDLWS Execution status	2:0	QDLWS 执行状态 3'b000: Idle 3'b001: NAK 3'b010: In Progress 3'b011: Timeout 3'b100: Successful Done	RO_DE0_EO
Reserved	31:3	Reserved	RO_DE0_EO

**D.6.17 PORT\_CAP16\_RSVD**

Reserved

**D.6.18 PORT\_CAP17\_RSVD**

Reserved

**D.6.19 PORT\_CAP18\_RSVD**

Reserved

**D.6.20 PORT\_CAP19\_RSVD**

Reserved

**D.6.21 PORT\_CAP20\_LMSM\_ST****D.6.21.1 Slice Header****表 D-213 LMSM\_ST Slice Header**

Field Name	Value
Slice Version	0x0
Slice Used Size	0x5

**D.6.21.2 Capability**

Start Offset Address:0x1

**表 D-214 LMSM\_ST Capability 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

**D.6.21.3 Control 1**

Start Offset Address:0x2

**表 D-215 LMSM\_ST Control 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
LSSM State Timeout Enable	31:0	<p>指示是否启用 LSSM 状态超时，值为 1 表示启用超时功能。</p> <p>Bit Location 映射对应 LMSM 状态：</p> <ul style="list-style-type: none"> <li>bit0: Probe.Wait</li> <li>bit1: Probe.Confirm</li> <li>bit2: RXEQ_Optimize</li> <li>bit3: Discovery.Active</li> <li>bit4: Discovery.Confirm</li> <li>bit5: Config.Active</li> <li>bit6: Config.Check</li> <li>bit7: Config.Confirm</li> <li>bit8: Send_NullBlock</li> <li>bit9: Reserved</li> <li>bit10: Retrain.Active</li> <li>bit11: Retrain.Confirm</li> <li>bit12: Reserved</li> <li>bit13: Change_Speed</li> <li>bit14: EQ.Coarse_Active</li> <li>bit15: EQ.Active</li> <li>bit16: EQ.Passive</li> <li>bit17: EQ.Coarse_Confirm</li> <li>bit18-31: Reserved</li> </ul>	RW_DE0_EO

**D.6.21.4 State 1**

Start Offset Address:0x3

**表 D-216 LMSM\_ST State 1 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

### D.6.21.5 State 2

Start Offset Address:0x4

**表 D-217 LMSM\_ST State 2 寄存器**

Field Name	Bit Location	Register Description	Attribute
Reserved	31:0	Reserved	RO_DE0_EO

## D.6.22 PORT\_CAP21\_PORT\_ERR\_RECORD

### D.6.22.1 Slice Header

**表 D-218 PORT\_ERR\_RECORD Slice Header**

Field Name	Value
Slice Version	0x0
Slice Used Size	0x1E

设备发生 Port 级错误时，将 Error 状态记录在 PORT\_ERR\_RECORD 寄存器中。每个 Port 存在一份该寄存器组。在该寄存器组中，地址 0x10 之前的寄存器为公共寄存器，0x10~0x1F 的寄存器是 Port 级错误。其整体结构如下：

注 1：此处描述的地址的单位是 4 Byte，不是 1 Byte。与配置空间整体一致。

注 2：PORT ERR RECORD 与 DEVICE ERR RECORD 的错误状态寄存器保持一致，但并非所有的错误都会在 PORT ERR RECORD 上报，具体在哪个寄存器组上报详见 10.6.2.2.4 C 类错误的上报描述。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address
Slice Header																												0x00				
Reserved																											0x01~0x11					
Port Uncorrectable Error Status																											0x12					
Port Uncorrectable Error Mask																											0x13					
Port Uncorrectable Error Severity																											0x14					
Port Correactable Error Status																											0x15					
Port Correactable Error Mask																											0x16					
Port Correactable Error Severity																											0x17					
Reserved																											0x18					
Reserved																											0x19					
Reserved																											0x1A					
Reserved																											0x1B					
Reserved																											0x1C					
Reserved																											0x1D					
Reserved																											0x1E					

**图 D-8 PORT\_ERR\_RECORD 结构示意图**

**D.6.22.2 Port Uncorrectable Error Status****表 D-219 Port Uncorrectable Error Status Register**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Value</b>
RSVD	19:0	预留给 20 种 CE 错误升级为 NUFE 错误的位置。	RW1CS_DE0_EO	0x12	0x0
DL Protocol Error Status	20	接收的 ACK 偏差超过最大值错误。 此错误被配置为 NUFE 时，此比特才生效。	RW1CS_DE0_EO	0x12	0x0
DL Replay Abort Error Status	21	重传过程发生异常，重传机制无法恢复错误状态。 此错误被配置为 NUFE 错误时，此比特才生效。	RW1CS_DE0_EO	0x12	0x0
RSVD	22	Reserved	RO	0x12	0x0
Packet Length Error Status	23	接收方向收到的包长度检查异常错误状态。 此错误被配置为 NUFE 错误时，此比特才生效。	RW1CS_DE0_EO	0x12	0x0
RSVD	24	Reserved	RO	0x12	0x0
RSVD	25	Reserved	RO	0x12	0x0
RSVD	26	Reserved	RO	0x12	0x0
ICRC Check Error Status	27	接收方向 ICRC 检查错误状态。 此错误被配置为 NUFE 错误时，此比特才生效。	RW1CS_DE0_EO	0x12	0x0
Receive Buffer Overflow	28	接收端 Receive Buffer 溢出错误。	RW1CS_DE0_EO	0x12	0x0
Flow Control Overflow	29	发送端接收 CRD 后超出初始化信用证，流控溢出错误。	RW1CS_DE0_EO	0x12	0x0
RSVD	31:30	Reserved	RO	0x12	0x0

**D.6.22.3 Port Uncorrectable Error Mask****表 D-220 Port Uncorrectable Error Mask Register**

<b>Field Name</b>	<b>Bit Location</b>	<b>Description</b>	<b>Attribute</b>	<b>Start Offset Address</b>	<b>Default Value</b>
RSVD	19:0	预留给 20 种 CE 错误升级为 NUFE 错误的位置。	RO	0x14	0x0
DL Protocol Error Mask	20	接收的 ACK 偏差超过最大值错误屏蔽。 此错误被配置为 NUFE 错误时，此比特才生效。	RW_DE0_EO	0x14	0x0
DL Replay Abort Error Mask	21	重传过程发生异常，重传机制无法恢复错误屏蔽。 此错误被配置为 NUFE 错误时，此比特才生效。	RW_DE0_EO	0x14	0x0
RSVD	22	Reserved	RO	0x14	0x0
Packet Length Error Mask	23	接收方向收到的包长度检查异常错误屏蔽。 此错误被配置为 NUFE 错误时，此比特才生效。	RW_DE0_EO	0x14	0x0
RSVD	24	Reserved	RO	0x14	0x0
RSVD	25	Reserved	RO	0x14	0x0
RSVD	26	Reserved	RO	0x14	0x0
ICRC Check Error Mask	27	接收方向 ICRC 检查错误屏蔽。 此错误被配置为 NUFE 错误时，此比特才生效。	RW_DE0_EO	0x14	0x0
Receive Buffer Overflow Mask	28	接收端 Receive Buffer 溢出错误屏蔽。	RW_DE0_EO	0x14	0x0
Flow Control Overflow Mask	29	发送端接收 CRD 后超出初始化信用证，流控溢出错误屏蔽。	RW_DE0_EO	0x14	0x0
RSVD	31:30	Reserved	RO	0x14	0x0

**D.6.22.4 Port Uncorrectable Error Severity**

表 D-221 Port Uncorrectable Error Severity Register

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Uncorrectable Error Severity	63:0	<p>控制 Device Uncorrectable Error Status 中每 bit 错误上报 Fatal Uncorrectable Error, 还是 Non-Fatal Uncorrectable Error。</p> <p>1'b1: 上报 Fatal Uncorrectable Error;</p> <p>1'b0: 上报 Non-Fatal Uncorrectable Error;</p> <p>仅 Bit[29:28]有效, 其它 bit 不允许配置。</p> <p>Bit[27:20]对应指示的错误从 CE 升级为 NFUE 时由 Correctable Error Severity 寄存器配置。</p>	RW_DE0_EO	0x16	0x0

**D.6.22.5 Port Correctable Error Status**

表 D-222 Port Correctable Error Status Register

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	0	Reserved	RO	0x18	0x0
Block Align Unlock	1	Block 失去锁定。	RW1CS_DE0_EO	0x18	0x0
Elasticity Buffer Overflow/Underflow	2	弹性 Buffer 上溢或下溢。	RW1CS_DE0_EO	0x18	0x0
Loss of Lane-to-Lane Deskew	3	Lane 间 Deskew 失锁。	RW1CS_DE0_EO	0x18	0x0
AMCTL decode Error	4	AMCTL 解码错误。	RW1CS_DE0_EO	0x18	0x0
LTB CRC Error	5	LTB CRC 校验错误。	RW1CS_DE0_EO	0x18	0x0
Link speed reduce Error	6	非预期的降速事件。	RW1CS_DE0_EO	0x18	0x0
Link width reduce Error	7	非预期的降 Lane 事件。	RW1CS_DE0_EO	0x18	0x0
Equalization	8	Equalization 粗调阶	RW1CS_DE0_EO	0x18	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Coarse tune Timeout		段超时。			
Equalization Finetune Timeout	9	EQ.Passive 状态或 EQ.Active 状态超时	RW1CS_DE0_EO	0x18	0x0
Link width negotiation Timeout	10	Link width 协商超时。	RW1CS_DE0_EO	0x18	0x0
LMSM Discovery Timeout	11	LMSM Discovery 阶段超时。	RW1CS_DE0_EO	0x18	0x0
RSVD	13:12	Reserved	RO	0x18	0x0
QDLWS timeout	14	动态升降 Lane 超时。	RW1CS_DE0_EO	0x18	0x0
Unsupported LW Switch REQ	15	收到不支持的 Link width 切换请求。	RW1CS_DE0_EO	0x18	0x0
RSVD	16	Reserved	RO	0x18	0x0
LMSM Retrain state Timeout	17	LMSM Retrain state 超时。	RW1CS_DE0_EO	0x18	0x0
DL Retry ACK Timeout	18	DL 发送端等待 Retry ACK 超时。	RW1CS_DE0_EO	0x18	0x0
DL Retry Rollover	19	DL 重传相同 RcvPtr 的 Packet 超次。	RW1CS_DE0_EO	0x18	0x0
DL Protocol Error	20	数据链路层协议错误。 此错误被配置为 CE 错误时，此比特才生效。	RW1CS_DE0_EO	0x18	0x0
DL Retry Error	21	DL 重传状态机进入 Error 状态。 此错误被配置为 CE 错误时，此比特才生效。	RW1CS_DE0_EO	0x18	0x0
RSVD	22	Reserved	RO	0x18	0x0
Packet Length Error	23	包的长度检查出错。 此错误被配置为 CE 错误时，此比特才生效。	RW1CS_DE0_EO	0x18	0x0
RSVD	26:24	Reserved	RO	0x18	0x0
ICRC Check Error	27	接收方向 ICRC 校验	RW1CS_DE0_EO	0x18	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
		错误。 此错误被配置为 CE 错误时，此比特才 生效。			
RSVD	31:28	Reserved	RO	0x18	0x0

#### D.6.22.6 Port Correctable Error Mask

表 D-223 Port Correctable Error Mask Register

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	0	Reserved	RO	0x1A	0x0
Block Align Unlock Mask	1	Block 失去锁定错误屏蔽。	RW_DE0_EO	0x1A	0x0
Elasticity Buffer Overflow/Underflow Mask	2	弹性 Buffer 上溢出或下溢出错误屏蔽。	RW_DE0_EO	0x1A	0x0
Loss of Lane-to-Lane Deskew Mask	3	Lane 间 Deskew 失锁错误屏蔽。	RW_DE0_EO	0x1A	0x0
AMCTL decode Error Mask	4	AMCTL 解码错误屏蔽。	RW_DE0_EO	0x1A	0x0
LTB CRC Error Mask	5	LTB CRC 校验错误屏蔽。	RW_DE0_EO	0x1A	0x0
Link speed reduce Error Mask	6	非预期的降速事件错误屏蔽。	RW_DE0_EO	0x1A	0x0
Link width reduce Error Mask	7	非预期的降 Lane 事件错误屏蔽。	RW_DE0_EO	0x1A	0x0
Equalization Coarsestune Timeout Mask	8	Equalization 粗调阶段超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
Equalization Finetune Timeout Mask	9	EQ.Passive 状态或 EQ.Active 状态超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
Link width negotiation Timeout Mask	10	Link 宽度协商超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
LMSM Discovery State Timeout Mask	11	链路发现阶段超时错误屏蔽。	RW_DE0_EO	0x1A	0x0

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
RSVD	12	Reserved	RO	0x1A	0x0
RSVD	13	Reserved	RO	0x1A	0x0
QDLWS timeout Mask	14	动态升降 lane 超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
Unsupport LW Switch REQ Mask	15	收到不支持的 Lane 宽切换请求错误屏蔽。	RW_DE0_EO	0x1A	0x0
RSVD	16	Reserved	RW_DE0_EO	0x1A	0x0
LMSM Retrain state Timeout Mask	17	LMSM Retrain state 超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
DL Retry ACK Timeout Mask	18	DL 发送端等待 Retry ACK 超时错误屏蔽。	RW_DE0_EO	0x1A	0x0
DL Retry Rollover Mask	19	DL 重传相同 RcvPtr 的 Packet 超次错误屏蔽。	RW_DE0_EO	0x1A	0x0
DL Protocol Error Mask	20	数据链路层协议错误屏蔽。 此错误被配置为 CE 错误时，此比特才生效。	RW_DE0_EO	0x1A	0x1
DL Retry Error Mask	21	重传过程发生异常，重传机制无法恢复错误屏蔽。 此错误被配置为 CE 错误时，此比特才生效。	RW_DE0_EO	0x1A	0x1
RSVD	22	Reserved	RO	0x1A	0x0
Packet Length Error Mask	23	包的长度检查出错错误屏蔽。 此错误被配置为 CE 错误时，此比特才生效。	RW_DE0_EO	0x1A	0x0
RSVD	24	Reserved	RO	0x1A	0x0
RSVD	25	Reserved	RO	0x1A	0x0
RSVD	26	Reserved	RO	0x1A	0x0
ICRC Check Error Mask	27	接收方向 ICRC 检查错误屏蔽。 此错误被配置为 CE 错误时，此比特才生效。	RW_DE0_EO	0x1A	0x0
RSVD	31:28	Reserved	RO	0x1A	0x0

### D.6.22.7 Port Correctable Error Severity

表 D-224 Port Correctable Error Severity Register

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Correctable Error Severity	31:0	控制 Correctable Status 中每 bit 错误上报 Correctable Error, 还是 Non-Fatal Error。 1'b1: 上报 Correctable Error; 1'b0: 上报 Non-Fatal Uncorrectable Error; 当前仅 Bit[27:20]生效，其它 Bit 不允许配置。	RW_DE0_EO	0x1C	0xFCFFFFFF

### D.6.23 PORT\_CAP240\_VD

表 D-225 PORT\_CAP240\_VD Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	Vendor 自定义

预留 PORT\_CAP240\_XX 到 PORT\_CAP255\_XX 为 Vendor define 的 CAP。

## D.7 CFG0\_ROUTE\_TABLE

### D.7.1.1 一般要求

每个 Routing Table Entry 占用 Y 个 4 字节, 其计算公式为  $Y = [(X-1) >> 5 + 1]$ , X 为 CFG0\_BASIC.Total Number of Ports 寄存器的值。Routing Table Entry Z 描述的是 Destination Network Address 等于 Z 时可以从哪些端口发出的信息,  $0 \leq Z \leq N$ , N 为 Max Index of Route Table Entry 寄存器值。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset Address																
Slice Header																													0x0																			
Reserved										Max Index of Routing Table Entry																			0x1																			
Reserved																													0x2																			
Reserved																													...																			
Default Routing Table																													0x10																			
Routing_Table_Entry0																													...																			
Routing_Table_Entry1																													0x10+(0+1)*Y																			
Routing_Table_Entry2																													0x10+(1+1)*Y-1																			
...																													0x10+(2+1)*Y																			
Routing_Table_EntryN																													0x10+(3+1)*Y-1																			
...																													0x10+(N+1)*Y																			
...																													0x10+(N+1+1)*Y-1																			

图 D-9 CFG0\_ROUTE\_TABLE 整体结构

表 D-226 CFG0 ROUTE TABLE Slice Header

Field Name	Value
Slice Version	0x0
Slice Used Size	$0x10+(N+1+1)*Y$

### D.7.1.2 ROUTE\_TABLE\_CAP

超出声明数量范围外的 Routing Table Entry，其寄存器属性将变成 RO 属性。

表 D-227 ROUTE\_TABLE\_CAP

Field Name	Bit Location	Description	Attribute	Start Offset Address	Default Value
Max Index of Routing Table Entry	15:0	指示支持 Routing Table Entry 的最大索引 注：Entry 数量等于 N+1，N 为该寄存器值。	RO_DE0_EO	0x1	0x0
Exact Route Supported	16	是否支持精确路由 0：不支持 1：支持	RO_DE0_EO	0x1	0x0

### D.7.1.3 Exact Route Enable

表 D-228 Exact Route Enable

Field Name	Bit Location	Description	Attribute	Start Address	Offset	Default Value
Exact Route Enable	0	启用精确路由： 0: 不启用； 1: 启用；	RW_DE0_EO	0x2		0x0

### D.7.1.4 Default Routing Table

表 D-229 Default Routing Table

Field Name	Bit Location	Description	Attribute	Start Address	Offset	Default Value
Default Routing Table	Y*32-1:0	每 bit 的含义为包是否可以从对应的 Port 发出 0: 不可以从对应 Port 发出； 1: 可以从对应 Port 发出；	RW_DE0_EO	0x10		0x0

### D.7.1.5 Routing Table Entry

表 D-230 Routing Table Entry

Field Name	Bit Location	Description	Attribute	Start Address	Offset	Default Value
Routing Table EntryZ	Y*32-1:0	表示每 bit 的含义为 DCNA 为 Z 的包是否可以从对应的 Port 发出。 0: 不可以从对应 Port 发出； 1: 可以从对应 Port 发出；	RW_DE0_EO	0x10 + ( Z+1 ) *Y		0x0

注：超出 Total Number of Ports 寄存器值对应范围的 bit，寄存器属性为 RO。

# 附录 E 以太互通

UB 网络可支持通过以下两种方式和以太网互通：

1. UBPU 通过 UBoE 直接接入以太网络；
2. UB2E Switch 实现 UB 链路和以太链路之间的转换：
  - (1) UB2E Switch 将 UB 链路层转换为以太网链路层时，应保留 IP 网络层及以上信息，可选择将 NPI 转换到 VLAN ID。
  - (2) UB2E Switch 将以太网链路层转换为 UB 链路层时，应支持添加 UB 网络扩展头。
  - (3) UB2E Switch 可支持 FECN 与 ECN 拥塞标记转换，打通 UB 域和以太域的拥塞控制。

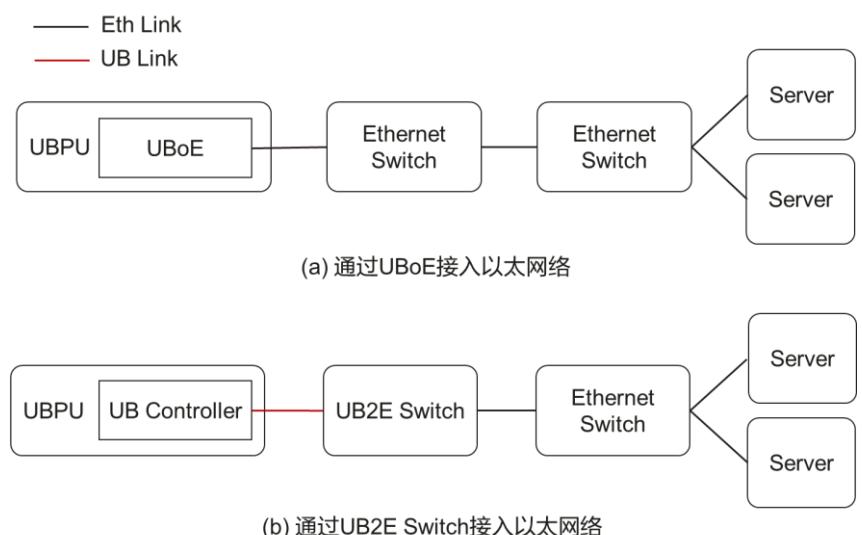


图 E-1 以太互通

以上两种方式在以太网上均用 UBoE 数据包承载。UBoE 数据包是指 UB 事务层和传输层承载在以太网上的数据包格式。

DMAC (6B)	SMAC (6B)	VLAN	EthType (2B)	IP	UDP (sport, dport=4792)	TPH	TAH	Payload
--------------	--------------	------	-----------------	----	----------------------------	-----	-----	---------

图 E-2 UBoE 数据包格式

以太网设备应开启 Jumbo 帧功能，防止数据包长度超出最大值导致丢包。

# 附录 F 基于 UB 链路的网络管理

## F.1 适用场景

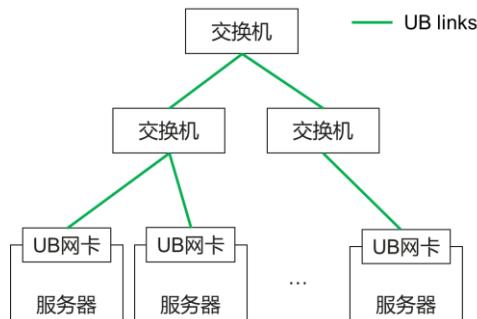


图 F-1 UB 支持传统服务器集群组网

在传统服务器集群组网场景，服务器通过 UB 网卡接入到交换机网络，若需要兼容以太/IP 网络的主流管理方式，可使用基于 UB 链路的网络管理协议进行管理。

UB 网卡的网络侧端口使用 UB 链路，UB 链路可直接对接 IP 网络层（硬件类型编号 38，Unified Bus (UB)，IANA Address Resolution Protocol Parameters）。UB 网卡为软件栈提供网络接口设备，网络接口设备使用 GUID 作为硬件地址。

## F.2 管理协议

### F.2.1 基于 UB 链路的管理协议包格式

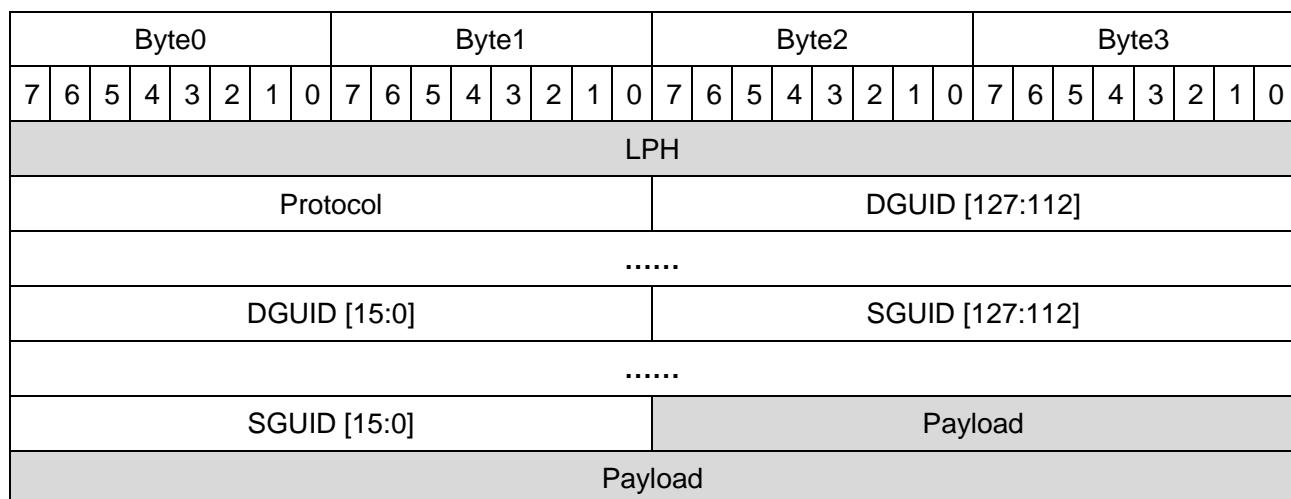


图 F-2 基于 UB 链路的管理协议包格式

表 F-1 管理协议包头字段

字段名	位宽(bit)	描述
Protocol	16	用于指示管理协议类型。
DGUID	128	Destination GUID，目的网络设备 GUID 或知名 GUID，用于指示接收数据包的网络设备身份。
SGUID	128	Source GUID，源网络设备 GUID，用于指示发送数据包的网络设备身份。

承载在 UB 链路上的管理协议，链路层包头中应将 LPH.CFG 设置为 5，不同管理协议的数据包采用 Protocol 域段区分，支持的管理协议如下：

表 F-2 管理协议

Protocol 域段	协议名称
0x0100	UB DHCPv4
0x0101	UB DHCPv6
0x0102	IP notify
0x0103	RSVD
0x0104	ULAP
0x0105	RSVD
0x0106	NPIC
0x0107-0x0109	RSVD
0x0110-0x0118	厂商自定义

## F.2.2 动态主机配置协议

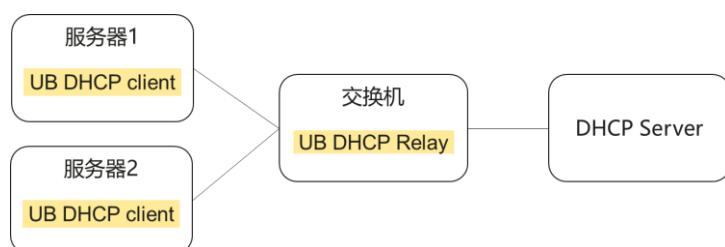


图 F-3 IP 地址动态管理

网络地址动态管理采用 UB DHCPv4/DHCPv6 协议实现，涉及 UB DHCP Client、UB DHCP Relay（可选）和 DHCP Server 三个组件。如上图所示，服务器通过 UB 网卡连接到交换机，服务器侧部署 UB DHCP Client，接入交换机上部署 UB DHCP Relay。主机动态申请 IP 地址时，通过 UB DHCP client 发起请求。UB DHCP Relay 协助完成 Client 与 DHCP Server 之间的协商。UB DHCP Relay 与 DHCP Server 之间可以使用 UB 网络或以太网络连接。

UB DHCP 基于 DHCP 协议 ( IETF RFC2131, IETF RFC8415 ), 将 IP 数据包基于 UB 链路承载且明确了 DHCP PDU 部分字段的使用规则, 具体如下:

UB DHCP client 与直连的 relay/server 之间交互消息采用如下格式:

LPH	Protocol	<SGUID,DGUID>	IP Header	UDP Header	DHCP PDU
-----	----------	---------------	-----------	------------	----------

图 F-4 UB DHCP 封装格式

其中, 域段填充信息如下:

1. LPH.CFG = 5
2. Protocol = 0x0100 标识 DHCPv4; Protocol = 0x0101 标识 DHCPv6;
3. SGUID: 为发出 DHCP 的网络设备 GUID, 长度为 128 bits;
4. DGUID: 目的网络设备 GUID 或知名 GUID, 长度为 128 bits, 使用知名 GUID 时其低 16 bits 为 0x0100 ( DHCPv4 ) 或 0x0101 ( DHCPv6 ) 。

DHCP PDU 域段遵循以下规则:

1. htype: 应是 38 ( UB hardware type )
2. hlen: 应是 16
3. chaddr: 应是申请 IP 的网络设备的 GUID
4. client-identifier option: 可选使用, 内容应填入网络设备的 GUID

### F.2.3 IP 地址通告协议

IP 地址通告协议 ( IP notify ) 在配置网络设备的 IP 地址时触发, 将该网络设备的 IP 地址通告给直连的邻居节点, 邻居对收到的 IP notify 解析后可以在本地生成或刷新指向该 IP 的路由, 可将该路由信息发布到全网。



图 F-5 IP 通告示意图

IP notify 封装格式如下:

LPH	Protocol	<SGUID,DGUID>	IP notify PDU
-----	----------	---------------	---------------

图 F-6 IP notify 封装格式

1. LPH.CFG = 5;
2. Protocol = 0x0102 来标识 IP notify;
3. DGUID 长度为 128 bits, 知名 GUID, 其低 16 bits 为 0x0102。

IP notify 的 PDU 格式如下：

Byte 0								Byte 1								Byte 2								Byte 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
Version		RSVD		Mask								IP Address																			
IP Address (variable)																															

图 F-7 IP notify 的 PDU 格式

1. Version：指示 IP 地址类型，长度为 4 bits；1：IPv4 地址；2：IPv6 地址；Others：保留。
2. Mask：长度为 8 bits, 子网掩码 bit 位数。
3. IP Address：指示发出该数据包的网络设备已配置的 IP 地址，其长度与 Version 对应：IPv4 地址长度为 4 Bytes，IPv6 地址长度为 16 Bytes。

## F.2.4 UB 链路聚合协议

ULAP(UB Link Aggregation Protocol)是 UB 链路动态聚合与解聚合的协议，使得网络设备能够根据自身配置自动形成聚合链路并使用聚合链路收发数据。ULAP 基于 LACP 协议 ( IEEE 802.1AX )，将 LACP PDU 基于 UB 链路承载，封装格式如下：

LPH	Protocol	<SGUID, DGUID>	LACP PDU
-----	----------	----------------	----------

图 F-8 ULAP 封装格式

其中，域段填充信息如下：

1. LPH.CFG = 5
2. Protocol = 0x0104 标识 ULAP。
3. DGUID 长度为 128 bits，知名 GUID，其低 16 bits 为 0x0104。

当服务器与交换机 1 和 2 协商成功后，服务器将 bond ( mode = 4 ) 的多个网口配置成相同的 IP ( 即 bond\_IP )，并通过 IP notify 消息将其通告给交换机 1 和 2，交换机 1 和 2 生成对应的 bond 路由，再将路由发布到交换机 3。当 bond 链路发生故障时，再以相同的方式刷新交换机的 bond 路由。

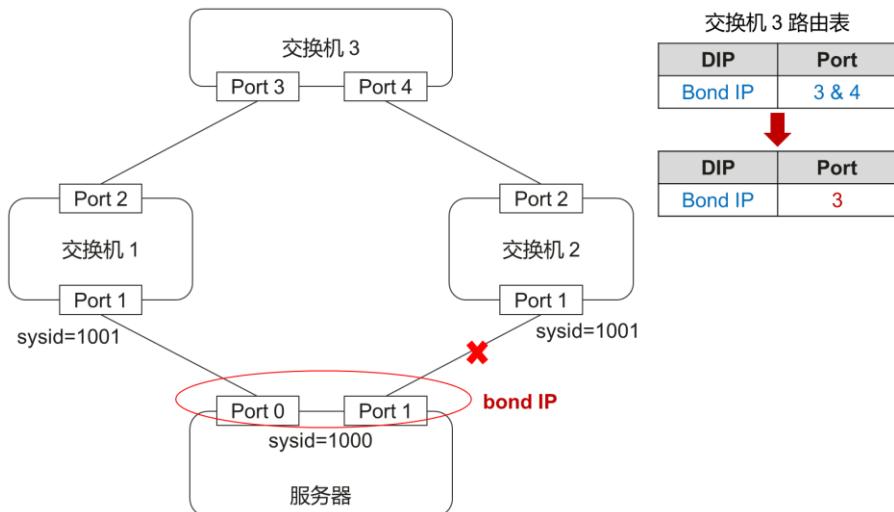


图 F-9 ULAP 工作机制

## F.2.5 网络分区配置协议

NPI 作为网络分区的标识，应保证配置过程的可信。服务器软件无法保证配置可信时，可以从网络侧配置 UB 网卡的 NPI。NPIC(NPI Configuration Protocol)支持带内配置 NPI，通过直连交换机实现配置下发。

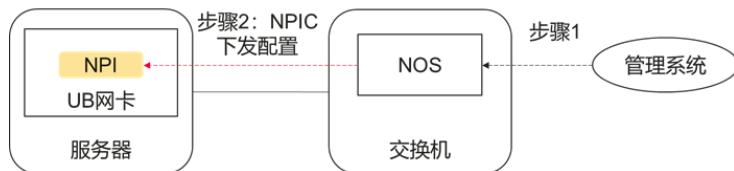


图 F-10 NPIC 配置过程

如上图所示，管理系统负责分配 UB 网卡的 NPI：

步骤 1：管理系统通过可信的带内或带外通道将 NPI 配置信息下发到对应的交换机，NPI 配置信息可选包含交换机端口 NPI 配置信息。

步骤 2：交换机将接收到的 UB 网卡的 NPI 配置信息通过 NPIC 下发到对应的 UB 网卡。由 UB 网卡执行配置，不经过任何服务器软件。

NPIC 封装格式如下：

LPH	Protocol	<SGUID, DGUID>	NPIC PDU
-----	----------	----------------	----------

图 F-11 NPIC 封装格式

其中，域段填充信息如下：

1. LPH.CFG = 5
2. Protocol = 0x0106 标识 NPIC。
3. DGUID 长度为 128 bits，知名 GUID，其低 16 bits 为 0x0106。

NPIC 的 PDU 格式如下：

Byte 0								Byte 1								Byte 2								Byte 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
Opcode								RSVD								Sum															
								GUID0																							
								GUID1																							
								GUID2																							
								GUID3																							
								NPI																							
								...																							

图 F-12 NPIC 封装格式

1. Opcode: 长度为 8 bits, 指示操作类型, 值为 1: 配置下发; 值为 2: 配置清除 ( NPI=0 ) ; 其它值: 保留。
2. Sum: 长度为 16 bits, 指示数据包中<GUID, NPI>对的数量。
3. GUID: 长度为 128 bits, 用于指示 UB 网卡。
4. NPI: 域段长度为 32 bits, 指示 NPI 值, 低 25 bits 有效。

## F.2.6 User Defined Protocol

LPH	Protocol	<SGUID, DGUID>	PDU
-----	----------	----------------	-----

图 F-13 用户自定义的封装格式

管理协议预留一段 Protocol 区间给用户自定义使用, 域段填充信息如下:

1. LPH.CFG = 5
2. Protocol 用户自定义的范围为 0x109-0x118。

## 附录 G 设备热插拔

## G.1 一般要求

在服务器内，UB 设备可支持热插入和热拔出操作。热插入是在系统工作期间，可将设备动态添加到系统中使用。热拔出允许在系统工作期间，将不再使用或出现错误的设备从系统中动态移除。

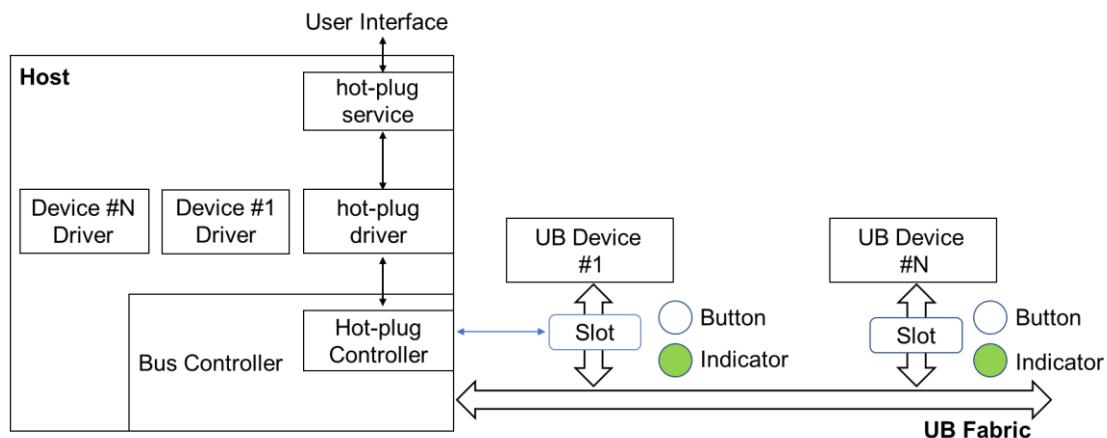


图 G-1 UB 系统热插拔软硬件

如上图所示，在支持热插拔的服务器中，应包括用户接口、热插拔服务、热插拔驱动、设备驱动、槽位、按键、指示灯等软硬件组件。支持热插拔的总线控制器应在 CFG0 中实现热插拔功能寄存器组，包括槽位描述、状态显示、热插拔控制等。槽位对应的 UB 设备端口链路状态变化后，应上报事件给热插拔控制器。

## G.2 热插拔组件

如下表，描述了支持设备热插拔应实现的软硬件组件。

表 G-1 热插拔组件

组件	功能描述
指示灯	指示该槽位的状态，具体见下文指示灯描述
按键	通知系统该设备需要移除
电源管理模块	系统软件控制电源管理模块，可以对槽位进行下电和上电操作
槽位	热插拔操作以槽位为单位，提供热插拔能力。
电源控制器	提供软件控制槽位电源的能力，并且可以检测槽位电源状态是否正常。

组件	功能描述
用户接口	系统软件通过软硬件接口获取槽位状态，给用户呈现设备状态等信息。

1. 指示灯
2. 每个槽位存在两个指示灯：电源状态指示和设备工作状态指示，每个指示灯都有三个状态，亮，灭和闪烁。

表 G-2 设备工作状态指示灯

指示灯状态	描述
灭	正常使用状态，此时不允许拔出设备。
亮	设备停止使用或者设备正在故障中，需要用户关注，必要时需要人工排查故障。
闪烁	当前正在被用户请求操作中，此时设备可以被安全的移除。

表 G-3 电源状态指示灯

指示灯状态	描述
灭	槽位下电状态，此时设备可以被插入或者拔出
亮	槽位上电状态，此时设备不允许插入或者拔出
闪烁	当前正在被用户请求操作中，此时设备不允许被插入或者拔出

### 3. 按键

支持热插拔时，应为每个槽位配置一个通知按键，通过按键实现热拔出或热插入操作通知。按键触发系统软件处理设备在位状态，当软件收到按键操作，需要判断当前槽位设备状态，如果设备目前在位且在工作中，触发软件进行热拔出操作。当槽位设备不在位或者槽位为下电状态时，触发软件进行热插入操作。

按键的具体实现不在协议中描述。

### 4. 电源管理模块

电源管理模块提供对槽位电源控制，通常由服务器单板的控制部件与系统软件配合控制槽位上电与下电。在槽位对应的设备需要热拔出时，宜先对槽位下电，再安全移除设备。当槽位没有设备时，系统软件可控制槽位下电，此时槽位不再工作。在槽位需要热插入设备时，宜在槽位下电状态下将设备连接到槽位，系统软件再启用槽位电源，随后开始初始化连接的设备。

### 5. 槽位

热插拔操作以槽位为单位进行。一个槽位可以包含一个或者多个连续编号的 Port，当槽位包含多个 Port 时，热插拔操作需要多个 Port 同时进行。设备支持的槽位数量以及每个槽位包含的 Port

数量与设备的物理实施形态相关。Entity 0 的 CFG0\_CAP2\_SHP 寄存器中包含了支持的槽位数量，以及每个槽位包含的 Port 信息。

#### 6. 电源控制器

电源控制器是热插拔组件中的可选组件，提供软件控制槽位电源状态的接口。电源控制器应检测槽位的电源状态。

#### 7. 用户接口

用户接口提供了用户处理热插拔操作的接口。在支持多个可热插拔槽位的系统中，用户接口允许用户独立地初始化每一个槽位。

## G.3 热拔出流程

当设备需要热拔出时，可通过如下流程进行：

### 1. 通知

用户通过按键通知系统软件，某个槽位设备需要热拔出。当按键被按下时，应发送对应槽位的按键消息到热插拔控制器，通知热插拔控制器有热插拔事件即将产生。槽位号可通过按键消息中 Slot Index 来获取。

### 2. 检查设备状态

在进行热拔出操作时，设备驱动程序应确保对应槽位的设备不再有任务需要执行。设备驱动程序可检查设备状态，是否还有未完成的任务，系统软件宜等待设备空闲后再进行热拔出操作。一个槽位支持多 Port 连接，设备驱动程序需要检查所有 Port 连接的设备都不存在未完成的任务时，才能认为设备为空闲状态。

### 3. 判断设备是否可安全移除

当系统软件确定设备空闲后，操作指示灯指示设备已停止使用，此时可卸载设备驱动。

端口设备在位状态发生变化，触发设备在位中断。

### 4. 移除设备，清理设备信息

系统软件处理设备在位中断，确定设备不在位后，回收设备资源，清除端口对应的路由信息。

完成设备资源回收后，系统软件可将槽位电源断开。至此，设备可安全拔出。

## G.4 热插入流程

当槽位需要热插入设备时，可通过如下流程进行：

### 1. 确认槽位下电，插入设备

设备热插入时，可通过槽位电源指示灯，确认槽位处于下电状态，再将设备插入到槽位上。

### 2. 按键操作，触发槽位上电

当槽位为下电状态时，用户可通过按键操作，触发槽位上电，系统软件修改槽位设备在位状态，触发设备在位中断。

3. 检查设备在位状态

系统软件收到设备在位中断后，检查对应槽位状态，确认设备在位。进入设备热插入处理流程。

4. 添加设备

系统软件处理设备在位中断，对新上电槽位进行设备扫描操作，添加设备，并为设备分配资源，下发路由信息。

5. 加载设备驱动

系统软件加载设备驱动，初始化设备。至此，设备可正常使用。

## G.5 热插拔事件

支持热插拔的槽位应支持如下事件处理：

1. 按键操作事件

当热插拔控制器接收到按键操作事件后，应根据热插拔消息发送控制寄存器指示，上报 Hot-plug event 到指定的目的端。

2. 设备在位检测事件

当热插拔控制器接收到设备在位检测状态发生变化的事件后，应根据热插拔消息发送控制寄存器指示，上报 Hot-plug event 到指定的目的端。

3. Port 链路状态检测事件

当热插拔控制器接收到链路状态发生变化的事件后，应根据热插拔消息发送控制寄存器指示，上报 Hot-plug event 到指定的目的端。

由于 Hot-plug event 可能在传输过程中丢失，导致上报失败，用户应确认接收到事件后作出插拔动作，否则需要重新触发热插拔。

对于正处于热插拔过程中的设备，Hot-plug event 与其它消息可能存在先后达到指定目的端的情况，目的端在处理其它消息时应做容错处理。目的端如接收到已被热拔出的设备发送的其它消息时应丢弃。

# 附录 H URPC 消息格式

## H.1 概述

本章列出 URPC 协议支持的消息格式，具体的协议设计见 8.5 节。

## H.2 URPC Function

Function[47:0]			
UBPU Class[11:0]	UBPU Subclass[11:0]	P[0]	Method[22:0]

图 H-1 URPC Function 域段

URPC 通过 Function 识别调用哪类 UBPU 的哪种函数，分为固定的公共函数和可部署的定制函数。公共函数（Public Method）在相同的 UBPU 类型（UBPU Class）和子类型（UBPU Subclass）中，具有统一的定义，包括函数、接口、参数等。定制函数（Customized Method）可根据场景自定义能力、接口等，URPC 协议不做约束，可以动态地在 Server/Worker 上部署和删除，供 Client 调用。

表 H-1 URPC Function 域段

域段名	位宽 ( Bits )	说明
UBPU Class	12	指示 UBPU 类型
UBPU Subclass	12	指示 UBPU 子类型
P[private]	1	<ul style="list-style-type: none"> <li>• 0: Method 是固定的公共函数（Public Method）；</li> <li>• 1: Method 是可部署的定制函数（Customized Method）；</li> </ul>
Method	23	指示 URPC 的调用函数

Client 可以通过预定义的 Method 实现对 Server/Worker 的公共函数/定制函数的查询、定制函数的部署、定制函数的删除等。

保留的 P[private] + Method：

- 公共函数：
  - 0b0000\_0000\_0000\_0000\_0000\_0000：查询 UBPU 支持的公共函数；
- 定制函数：
  - 0b1000\_0000\_0000\_0000\_0000\_0000：查询 UBPU 支持的定制函数；
  - 0b1000\_0000\_0000\_0000\_0000\_0001：部署新的定制函数；
  - 0b1000\_0000\_0000\_0000\_0000\_0010：删除已部署的定制函数；

-0b1000\_0000\_0000\_0000\_0000\_0011: 查询指定名称的 Method;

**表 H-2 存储领域 URPC 公共 Function**

UBPU Class	UBPU Subclass	P	Method
[47:36]	[35:24]	[23:23]	[22:0]
0x002	0x001	0	...

## H.3 URPC Message

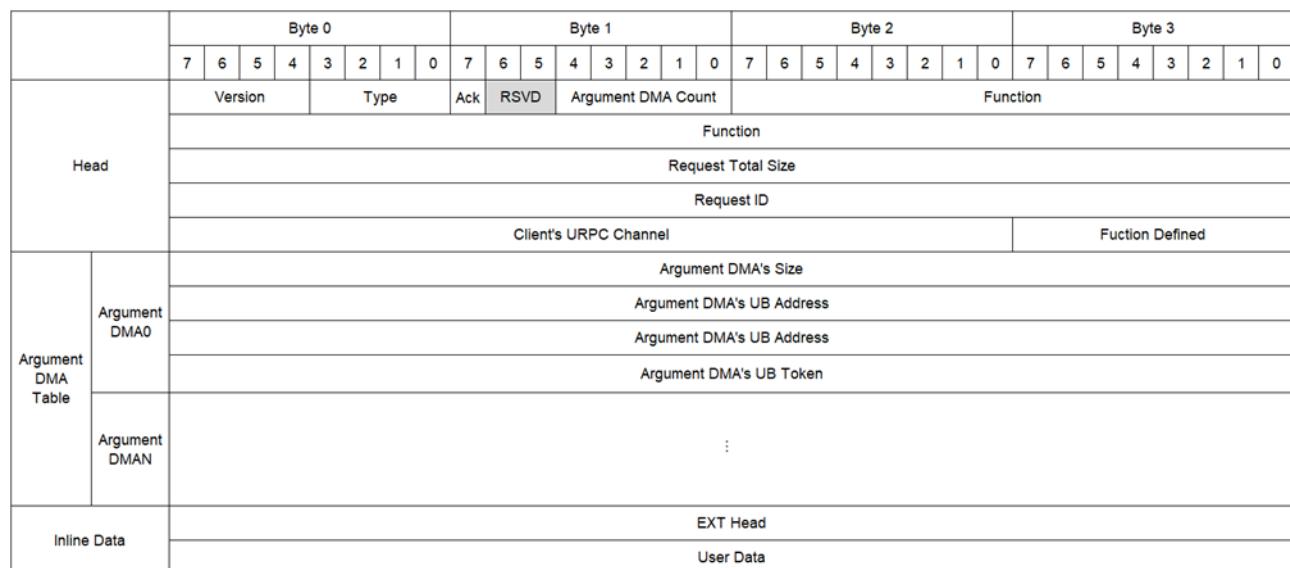
URPC Message 是 URPC Client 和 URPC Server 之间的交互报文，其格式定义如下。

### H.3.1 域段定义

**表 H-3 域段定义**

域段名	位宽 ( Bits )	说明
Type	4	指示 URPC Message 的类型格式： <ul style="list-style-type: none"> <li>• 0: URPC Request;</li> <li>• 1: URPC Ack;</li> <li>• 2: URPC Response;</li> <li>• 3: URPC Ack + URPC Response ( Message 格式与 URPC Response 相同 );</li> </ul>

### H.3.2 URPC Request



**图 H-2 URPC Request 域段**

表 H-4 URPC Request 域段

域段名	位宽 ( Bits )	说明
Version	4	URPC 版本号, 当前版本号为 1
Type	4	指示 URPC Message 的类型格式, 本报文填 0
Ack	1	指示 URPC 调用流程中是否需要 URPC Ack。Server 自主决定 URPC Ack 与 URPC Response 是否合并返回。 <ul style="list-style-type: none"><li>• 0: 不包含 URPC Ack;</li><li>• 1: 包含 URPC Ack;</li></ul>
Argument DMA Count	6	指示 Server 执行的 Argument DMA 的数量 当 Argument DMA Count == 0 时, Argument DMA Table 不存在
Function	48	指示 URPC 调用的 Function
Request Total Size	32	指示 URPC Request 的大小 ( 单位 Byte ), 包含协议头 ( Head ) 和全部参数分片的总大小, 不包含 Argument DMA Table 的大小
Request ID	32	唯一的 URPC 调用标识, 用于匹配 Request 与 Ack/Response
Client's URPC Channel	24	指示发送 URPC Request、接收 URPC Ack/Response 的 Client 侧 URPC Channel
Function Defined	8	指示函数面向的领域, 用于确定 URPC 扩展头的类型: <ul style="list-style-type: none"><li>• 0: 无扩展头;</li><li>• 1: 通用计算扩展;</li><li>• 2: 存储 PLOG 扩展;</li><li>• 其它: 待定义;</li></ul>
Argument DMA's Size	32	指示 Server 执行的 Argument DMA 的大小
Argument DMA's UB Address	64	指示 Server 执行的 Argument DMA 的 UB 地址
Argument DMA's UB Token	32	指示 Server 执行的 Argument DMA 的 UB Token
EXT Head	-	URPC Request 扩展协议, 具体域段自定义
User Data	-	在 URPC Request 中直接传输的用户数据
Inline Data	-	在 URPC Request 中直接传输的数据, 包含 URPC 扩展头 ( EXT Head ) 和用户数据 ( User Data )

### H.3.3 URPC Ack

	Byte 0								Byte 1								Byte 2								Byte 3																											
Head	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																				
	Version				Type				RSVD								Request ID Range																																			
	Request ID																																																			
	Client's URPC Channel																RSVD																																			

图 H-3 URPC Ack 域段

表 H-5 URPC Ack 域段

域段名	位宽( Bits )	说明
Version	4	URPC 版本号, 当前版本号为 1
Type	4	指示 URPC Message 的类型格式: <ul style="list-style-type: none"> <li>• 1: URPC Ack 报文独立;</li> <li>• 3: URPC Ack 和 URPC Response 报文合并 ( Message 格式与 URPC Response 相同 );</li> </ul>
Request ID Range	16	指示 URPC Ack/Response 包含的 Request ID 的范围。URPC Ack/Response 集体传输需满足条件: <ul style="list-style-type: none"> <li>• Request ID 连续;</li> <li>• URPC Channel 相同;</li> <li>• Status 相同;</li> </ul> Request ID Range 的默认值是 1, 仅包含当前 Request ID 的 URPC Response
Request ID	32	唯一的 URPC 调用标识, 用于匹配 Request 与 Ack/Response
Client's URPC Channel	24	指示发送 URPC Request、接收 URPC Ack/Response 的 Client 侧 URPC Channel

### H.3.4 URPC Response

	Byte 0								Byte 1								Byte 2								Byte 3																											
Head	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																				
	Version				Type				Status								Request ID Range																																			
	Request ID																Fuction Defined																																			
	Response Total Size																																																			
Return Data	Return Data Offset																Return Data Offset																																			
	:																																																			
	EXT Head																User Data																																			

图 H-4 URPC Response 域段

表 H-6 URPC Response 域段

域段名	位宽 ( Bits )	说明
Version	4	URPC 版本号，当前版本号为 1
Type	4	指示 URPC Message 的类型格式： <ul style="list-style-type: none"><li>• 2: URPC Response 报文独立；</li><li>• 3: URPC Ack 和 URPC Response 报文合并（Message 格式与 URPC Response 相同）；</li></ul>
Status	8	<ul style="list-style-type: none"><li>• 0: Worker 执行 URPC 完成；</li><li>• 1: Server 拒绝执行；</li><li>• 2: 调用 Function 不支持；</li><li>• 3: Server 的 Argument Buffer 不足；</li><li>• 4: URPC 调用超时；</li><li>• 5: Version 不匹配；</li><li>• 6: URPC 协议头错误；</li><li>• 其它：待定义；</li></ul>
Request ID Range	16	指示 URPC Ack/Response 包含的 Request ID 的范围。URPC Ack/Response 集体传输需满足条件： <ul style="list-style-type: none"><li>• Request ID 连续；</li><li>• URPC Channel 相同；</li><li>• Status 相同；</li></ul> Request ID Range 的默认值是 1，仅包含当前 Request ID 的 URPC Response
Request ID	32	唯一的 URPC 调用标识，用于匹配 Request 与 Ack/Response
Client's URPC Channel	24	指示发送 URPC Request、接收 URPC Ack/Response 的 Client 侧 URPC Channel
Function Defined	8	指示函数面向的领域，用于确定 URPC 扩展头的类型： <ul style="list-style-type: none"><li>• 0: 无扩展头；</li><li>• 1: 通用计算扩展；</li><li>• 2: 存储 PLOG 扩展；</li><li>• 其它：待定义；</li></ul>
Response Total Size	32	指示 URPC Response 的大小（单位 Byte），包含协议头（Head）和 Return Data 的总大小，不包含 Return Data Offset 的大小

域段名	位宽 ( Bits )	说明
Return Data Offset	32	<p>当 Return Data 存在时，指示集体传输 URPC Response 的每个 Request ID 的 Return Data 的偏移。集体传输 URPC Response 的第一个 URPC Request ( Request ID - Request ID Range + 1 ) 的 Return Data Offset 默认等于 0，并且省略。因此，Return Data Offset 的排列顺序依次从 ( Request ID - Request ID Range + 2 ) 到 Request ID</p> <ul style="list-style-type: none"> <li>• 当 Return Data 不存在时，Return Data Offset 不存在；</li> <li>• 当 Request ID Range == 1 时，Return Data Offset 不存在；</li> </ul>
EXT Head	-	URPC Response 扩展协议，具体域段自定义
User Data	-	在 URPC Response 中传输的用户数据
Return Data	-	在 URPC Response 中传输的响应数据，包含 URPC 扩展头 ( EXT Head ) 和用户数据 ( User Data )

# 附录 I 应用示例

## I.1 存储 PLOG

### I.1.1 概述

URPC 协议在存储领域应用，支持统一调用存储设备支持的函数，具备以下特征：

- 引用传参：存储数据作为参数，支持引用传递；
- 函数调用行为统一：支持跨主机经过 CPU 访问存储设备和直访存储设备两种方式，且对外接口统一。程序员可聚焦于业务实现，无需关心 Server 形态；

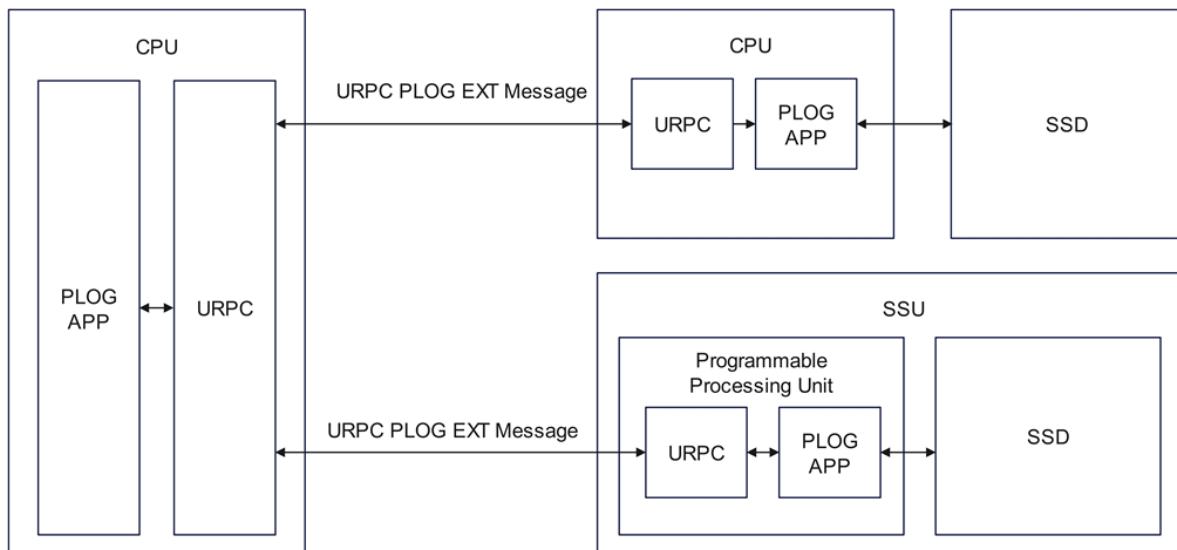


图 I-1 URPC 存储 PLOG 应用场景

存储 PLOG 应用将存储数据先封装 PLOG 协议，再调用统一的 URPC 接口访问存储，支持两种场景：

- CPU->CPU->SSD：CPU 发起的 URPC 请求，经过目的 CPU 处理后，再访问存储设备 SSD；
- CPU->SSU：CPU 发起的 URPC 请求，直接访问灵衢存储设备 SSU；

注：PLOG，分布式存储持久化协议；本示例应用使用的协议；

### I.1.2 URPC PLOG EXT Message

为了达成高性能访问存储的目标，在 URPC Message 的 EXT Head 区针对存储业务特征定义了扩展传输协议 URPC PLOG EXT Message。

Function Defined 值为 2 时，启用本扩展头。

### I.1.2.1 URPC PLOG EXT Request

	Byte 0								Byte 1								Byte 2								Byte 3																					
URPC Head	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														
URPC Plog EXT Head	URPC Request(20 bytes)																								User Data Offset																					
URPC Plog EXT Head	Version																User Head Offset																													
URPC Plog EXT Head	DMA's Count																Local Queue ID																													
URPC Plog EXT Head	Server's URPC Channel																									DMA Redirect	Early RSP	FC	RSVD																	
URPC Plog EXT Head	Local Queue TokenValue																									Security Head/RSVD(28 bytes)																				
User Head	User Head(192 bytes)																									Data DMA's UB Address																				
User Head	Data DMA's UB Address																									Data DMA's UB TokenID(L)																				
User Data or Data SGL	Data DMA's UB TokenID(M)				Data DMA's UB EID Index(L)				Data DMA's UB TokenID(H)				Data DMA's UB EID Index(H)				Data DMA's SGL Type																													
User Data or Data SGL	Data DMA's UB TokenValue																									RSVD	Data DMA's Flag																			
User Data or Data SGL	RSVD																									RSVD	RSVD																			
User Data or Data SGL	RSVD																									...																				

图 I-2 URPC PLOG EXT Request 域段

表 I-1 URPC PLOG EXT Request 域段

域段名	位宽 ( Bits )	说明
URPC Request	20 * 8	URPC 请求基础头，具体域段定义详见 H.3.2 节
Version	4	URPC PLOG EXT Request 版本号，当前版本号为 1
Data Mode	4	<p>数据传输模式：</p> <ul style="list-style-type: none"> <li>• 0: Send (Push), Client 使用 Send 语义将请求数据推送给 Server;</li> <li>• 1: Send (Push or Pull), Client 使用 Send 语义将请求数据推送给 Server; 当 Server 内存不足时，由 Server 使用 Read 语义反向从 Client 拉取请求数据;</li> <li>• 2: Send + Read, Client 使用 Send 语义将请求数据地址推送给 Server, 由 Server 使用 Read 语义反向从 Client 拉取请求数据;</li> <li>• 3: Send + Write, Client 使用 Send 语义将响应存放的数据地址推送给 Server, 由 Server 使用 Write 语义将响应返回 Client, 由 Server 确保先返回响应数据再返回响应结果;</li> <li>• 4: Send + Write_with_immediate, Client 使用 Send 语义将响应存放的数据地址推送给 Server, 由 Server 使用 Write_with_immediate 语义将响应返回 Client, 由 Client 确保先返回响应数据再返回响应结果;</li> <li>• 5: Send + Read + Write, Client 使用 Send 语义将请求数据地址和响应存放的数据地址推送给 Server, 由 Server 使用 Read 语义反向从 Client 拉取请求数据，使用 Write 语义将响应返回 Client;</li> </ul>

域段名	位宽 ( Bits )	说明
User Head Offset	8	用户头区域 ( User Head ) 相对扩展头起始的偏移
User Data Offset	16	用户数据区域相对扩展头起始的偏移 注：用户数据区域支持 User Data ( 数据 ) 和 Data SGL ( 数据 SGE 数组 ) 两种格式
DMA's Count	16	<ul style="list-style-type: none"> <li>• 0: 用户数据区域采用 User Data 格式；</li> <li>• &gt;0: 用户数据区区域采用 Data SGL 格式，指示 Data SGL 的 SGE 数量；</li> </ul>
Local Queue ID	16	本地 Queue 的 ID，进程内唯一，用于远端回复时作为索引关联本地 Queue 的上下文信息（例如：可用的 tjetty） 0 为无效值
Server's URPC Channel	24	Server 为 Client 的连接分配的 Channel ID
Early RSP	1	指示 Server 是否返回 RSP： <ul style="list-style-type: none"> <li>• 0: 指示 Server 需要返回 RSP；</li> <li>• 1: 指示 Server 无需返回 RSP；</li> </ul>
FC(Flow Control Flag)	2	流控标记： <ul style="list-style-type: none"> <li>• 0: schedule, 报文接受流控调度；</li> <li>• 1: unschedule: 报文不接受流控调度；</li> </ul>
DMA Redirect	1	开启 DMA 的目的端重定向到第三方（除 Client/Server 外）能力，具体目的端根据 Data DMA's UB EID Index 索引确定： <ul style="list-style-type: none"> <li>• 0: 关闭到第三方 UB 设备 DMA 的能力，默认 DMA 目的地是 Client 端 UB 设备；</li> <li>• 1: 开启到第三方 UB 设备 DMA 的能力；</li> </ul>
Local Queue TokenValue	32	本地 Queue 的 TokenValue 值，响应返回时用于校验 Queue 的合法性
Security Head	28 * 8	开启加解密和完整性校验时存放 IV (12 byte) + TAG (16 byte) 信息
User Head	192 * 8	用户 ( PLOG ) 自定义协议
Data DMA's UB Address	64	DMA 的数据地址
Data DMA's Size	32	DMA 的数据大小

域段名	位宽 ( Bits )	说明
Data DMA's SGL Type	8	SGL 数据类型： • 0x0B: Server 端处理本 SG； • 0x1B: Server 端不处理本 SG，跳过继续处理下一个 SG；
Data DMA's UB TokenID	20	数据 DMA 的 TokenID，用于远端访问内存的合法性校验
Data DMA's UB EID Index	12	数据 DMA 的目的 UB EID 索引，用于支持 DMA 到 Client/Server 外的第三方 UB 设备
Data DMA's UB TokenValue	32	数据 DMA 的 TokenValue，用于远端访问内存的合法性校验
Data DMA's Flag	4	数据 DMA 的属性标记： • 0: Read，用于 Server 向 Client 读此地址的数据； • 1: Write，用于 Server 向 Client 写数据到此地址； 约束：不支持 Read 和 Write 交叉填写，Read 的 SGE 必须在 Write 的 SGE 前面

### I.1.2.2 URPC PLOG EXT Response

	Byte 0								Byte 1								Byte 2								Byte 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											
URPC Head	URPC Response(16 bytes)																																										
URPC Plog EXT Head	Version				Data Mode				User Head Offset				User Data Offset																RSVD														
	DMA's Count																RSVD																										
	Server's URPC Channel																										RSVD																
	RSVD(8 bytes)																																										
User Head	User Head																																										
User Data	User Data																																										

图 I-3 URPC PLOG EXT Response 域段

表 I-2 URPC PLOG EXT Response 域段

域段名	位宽 ( Bits )	说明
URPC Response	16 * 8	URPC 响应基础头，具体域段定义详见 H.3.4 节
Version	4	URPC PLOG EXT Response 版本号，当前版本号为 1
Data Mode	4	数据传输模式： • 0: Send (Push)，Client 使用 Send 语义将请求数据推送给 Server； • 1: Send (Push or Pull)，Client 使用 Send 语义将请求数据推送给 Server；当 Server 内存不足时，由 Server 使

字段名	位宽 ( Bits )	说明
		<p>用 Read 语义反向从 Client 拉取请求数据；</p> <ul style="list-style-type: none"> <li>• 2: Send + Read, Client 使用 Send 语义将请求数据地址推送给 Server, 由 Server 使用 Read 语义反向从 Client 拉取请求数据；</li> <li>• 3: Send + Write, Client 使用 Send 语义将响应存放的数据地址推送给 Server, 由 Server 使用 Write 语义将响应返回 Client, 由 Server 确保先返回响应数据再返回响应结果；</li> <li>• 4: Send + Write_with_immediate, Client 使用 Send 语义将响应存放的数据地址推送给 Server, 由 Server 使用 Write_with_immediate 语义将响应返回 Client, 由 Client 确保先返回响应数据再返回响应结果；</li> <li>• 5: Send + Read + Write, Client 使用 Send 语义将请求数据地址和响应存放的数据地址推送给 Server , 由 Server 使用 Read 语义反向从 Client 拉取请求数据, 使用 Wirte 语义将响应返回 Client；</li> </ul>
User Head Offset	8	用户头区域 ( User Head ) 相对扩展头起始的偏移
User Data Offset	16	用户数据区域 ( User Data ) 相对扩展头起始的偏移
Data DMA's Count	16	服务端返回客户端的数据分片数量 该字段仅在数据传输模式 4: Send + Write_with_immediate 下生效
Server's URPC Channel	24	Server 为 Client 的连接分配的 Channel ID
Security Head	28 * 8	开启加解密和完整性校验时存放 IV (12 byte) + TAG (16 byte)信息
User Head	-	用户 ( PLOG ) 自定义协议
User Data	-	返回时响应携带的用户数据

### I.1.3 Data Mode 说明

#### Data Mode0: Send(Push)

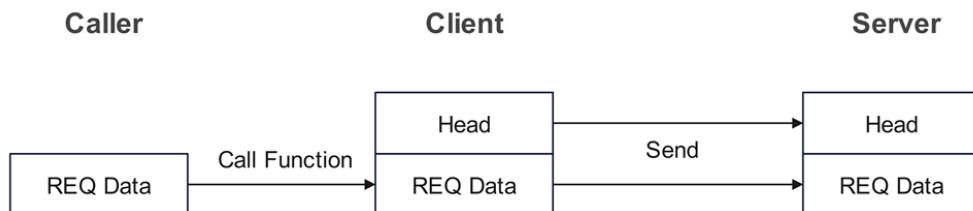


图 I-4 URPC PLOG 传输模式 0

- Caller 发起 Call Function，传递请求 Data；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、请求 Data 推送给 Server；

#### Data Mode1: Send(Push or Pull)

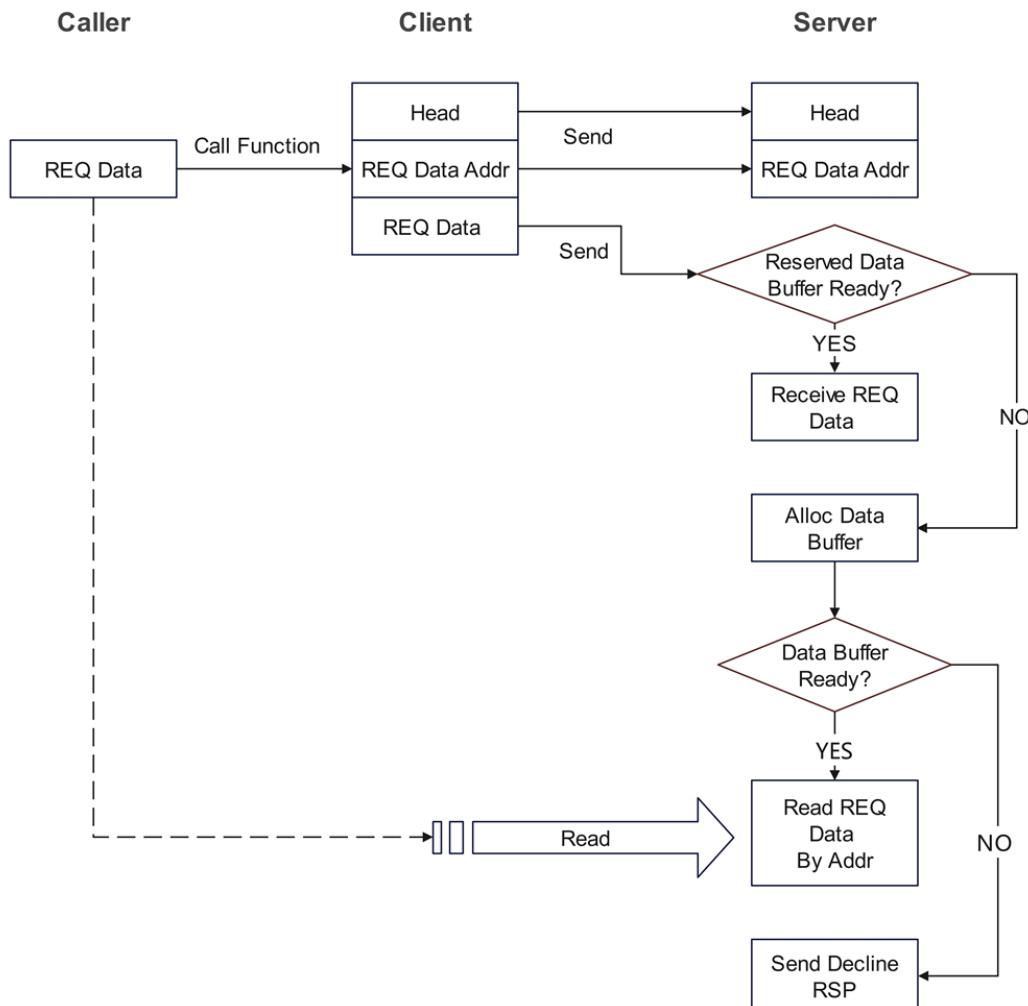


图 I-5 URPC PLOG 传输模式 1

- Caller 发起 Call Function，传递请求 Data；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、请求 Data 以及请求 Data 的地址推送给 Server；
- 当 Server 预留内存充足，Server 直接收取请求 Data；
- 当 Server 预留内存不足，Server 申请内存；
- 申请成功，使用 Read 语义基于 Data 的地址向 Client 拉取请求 Data；
- 申请失败，返回异常响应给 Server；

Data Mode2: Send+Read

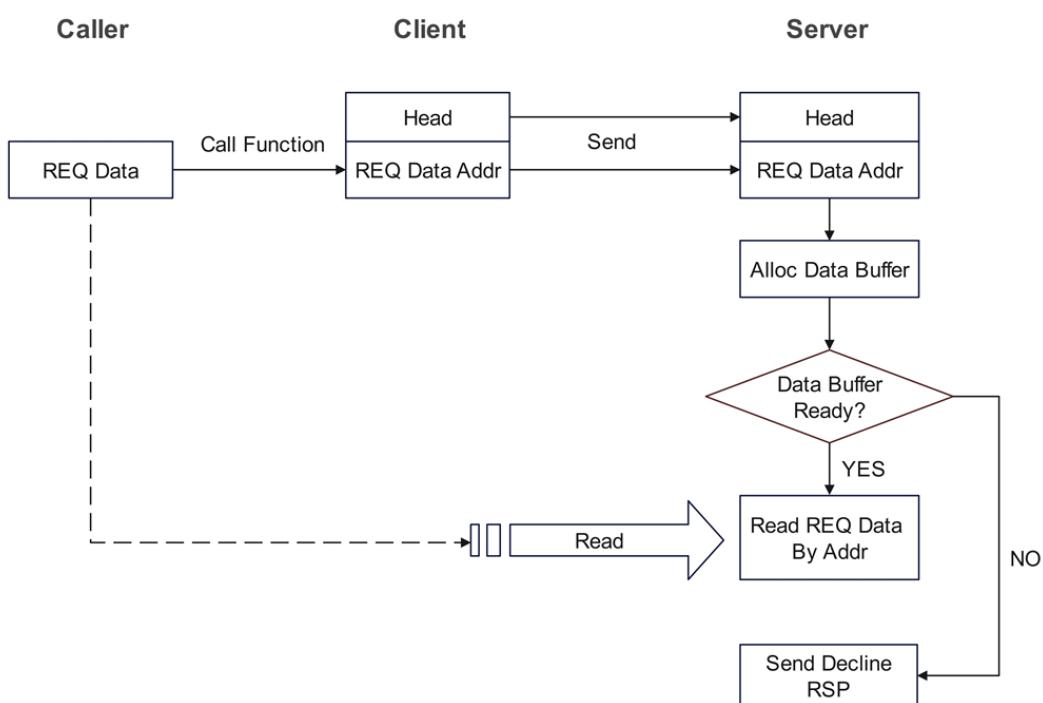


图 I-6 URPC PLOG 传输模式 2

- Caller 发起 Call Function，传递请求 Data；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、请求 Data 以及请求 Data 的地址推送给 Server；
- Server 申请内存；
- 申请成功，使用 Read 语义基于 Data 的地址向 Client 拉取请求 Data；
- 申请失败，返回异常响应给 Server；

**Data Mode3: Send+Write  
(Server Order-Preserving)**

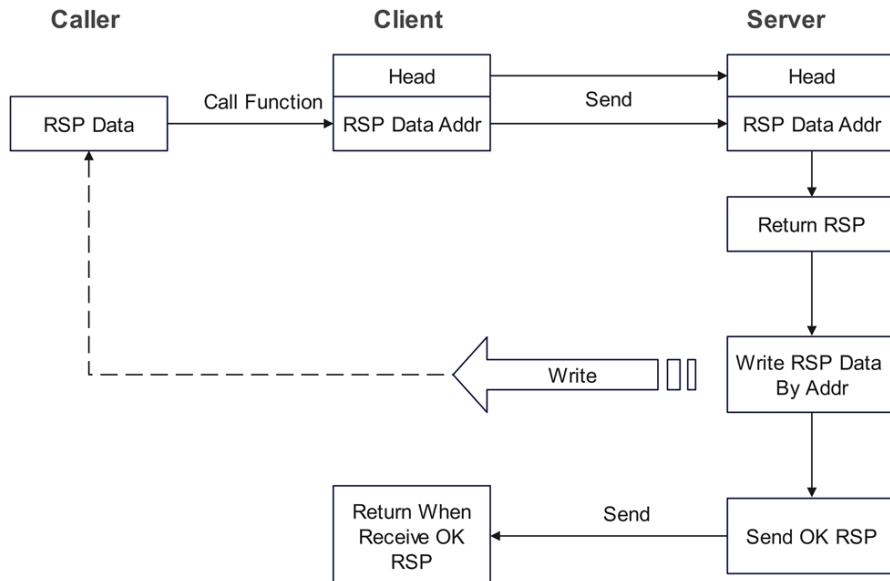


图 I-7 URPC PLOG 传输模式 3

- Caller 发起 Call Function，传递响应 Data 地址；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、响应 Data 的地址推送给 Server；
- Server 执行函数后，使用 Write 语义将响应数据写入响应 Data 的地址；
- Server 写响应 Data 成功后，发送响应结果给 Client；
- Client 收到响应结果后 Return Funcion；

**Data Mode4: Send+Write\_with\_immediate  
(Client Order-Preserving)**

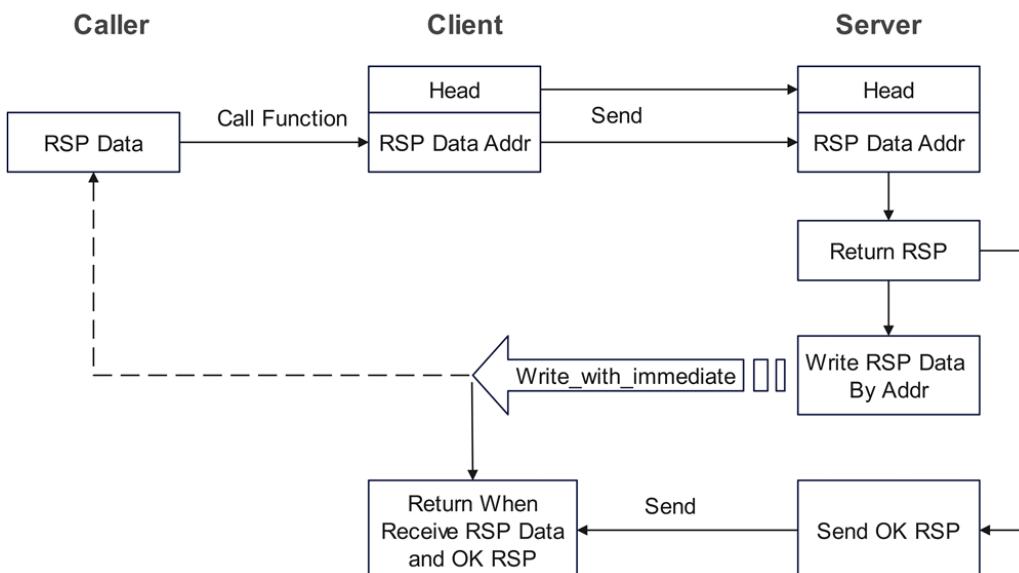


图 I-8 URPC PLOG 传输模式 4

- Caller 发起 Call Function，传递响应 Data 地址；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、响应 Data 的地址推送给 Server；
- Server 执行函数后，使用 Write\_with\_immediate 语义将响应数据写入响应 Data 的地址，同时发送响应结果给 Client；
- Client 确定响应数据以及响应结果都收到后 Return Funcion；

Data Mode5: Send+Read+Write

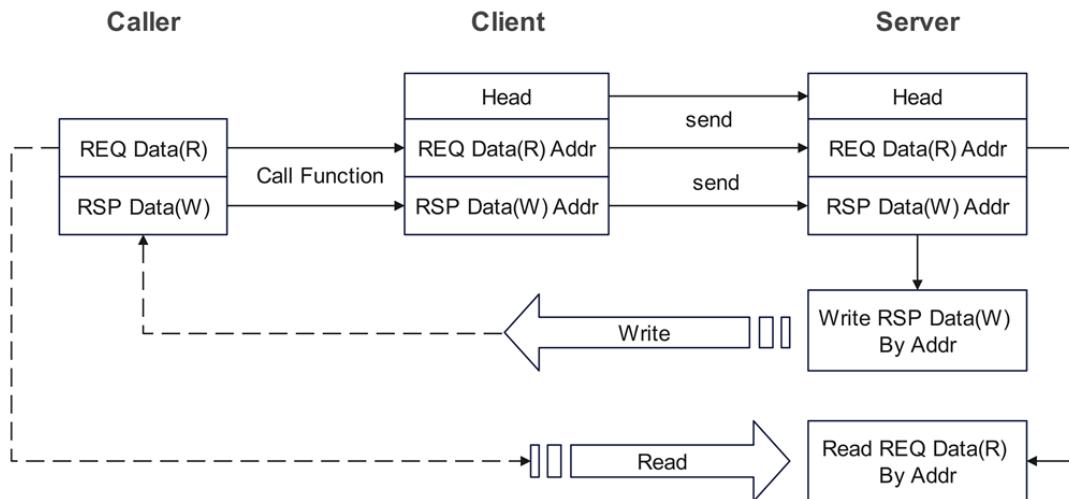


图 I-9 URPC PLOG 传输模式 5

- Caller 发起 Call Function，传递请求 Data(R)地址、响应 Data(W)地址；
- Client 封装 URRC 协议头，使用 Send 语义将 Head、请求 Data(R)地址、响应 Data(W)地址推送给 Server；
- Server 执行函数后，使用 Read 语义从 Client 拉取请求 Data 的地址；
- Server 执行函数后，使用 Write 语义将响应数据写入响应 Data 的地址；

表 I-3 Data Mode

Data Mode	推荐场景	特点
Mode0: Send(Push)	小 IO Request 传输	<ul style="list-style-type: none"> <li>• 数据直传，仅需 0.5 个 RTT 完成传输，时延低；</li> <li>• Server 需预占足量接收内存，内存占用大，利用率低；</li> </ul>
Mode1: Send(Push or Pull)	小 IO Request 传输	<ul style="list-style-type: none"> <li>• Server 内存足够时，数据直传，仅需 0.5 个 RTT 完成传输，时延低；</li> <li>• Push 和 Pull 模式切换处理流程复杂，且两种模式性能存在差距，造成访存时延抖动，0.5 或者 1.5 个 RTT 完成 Request 传输；</li> <li>• Server 无需预占大量内存用于接收 Request，</li> </ul>

Data Mode	推荐场景	特点
		不足时可灵活切换为 pull 模式；
Mode2: Send+Read	大 IO Request 传输	<ul style="list-style-type: none"> <li>需 1.5 个 RTT 完成 Request 传输；</li> <li>Server 无需预占大量内存用于接收 Request；</li> </ul>
Mode3: Send+Write (Server Order-Preserving)	大 IO Response 传输 网络传输距离短	<ul style="list-style-type: none"> <li>需 1.5 个 RTT 完成 Response 传输；</li> <li>Client 无需预占大量内存用于接收 Response；</li> <li>消耗 Server 端资源进行保序操作；</li> </ul>
Mode4: Send+ Write_with_immediate (Client Order-Preserving)	大 IO Response 传输 网络传输距离长	<ul style="list-style-type: none"> <li>需 0.5 个 RTT 完成 Response 传输；</li> <li>Client 无需预占大量内存用于接收 Response；</li> <li>消耗 Client 端资源进行保序操作；</li> </ul>
Mode5: Send+Read+Write	大 IO Request & Response 传输	<ul style="list-style-type: none"> <li>需 1.5 个 RTT 完成 Request 传输；</li> <li>需 1.5 个 RTT 完成 Response 传输；</li> <li>Server 无需预占大量内存用于接收 Request；</li> <li>Client 无需预占大量内存用于接收 Response；</li> </ul>

注：RTT ( Round-Trip Time ) 表示往返时间

# 图目录

图 2-1 UB Domain 系统组成 .....	13
图 2-2 跨 UB Domain 系统组成 .....	13
图 2-3 UB 协议栈 .....	15
图 3-1 物理层概览 .....	17
图 3-2 UB 链路组成示意 .....	18
图 3-3 物理层数据通路功能图 .....	20
图 3-4 RS FEC 编码电路 .....	21
图 3-5 非交织 FEC 模式下 Post-FEC 符号到 Lane 分发示意图 (x8) .....	23
图 3-6 交织模式下 Post-FEC 符号到 Lane 分发示意图 (x8) .....	24
图 3-7 eBCH-16 码字格式 .....	27
图 3-8 eBCH-16 全部可纠正的错误图示 .....	30
图 3-9 AMCTL 锁定参考流程图 .....	36
图 3-10 NRZ 模式下 Symbol 在 Lane 上的传输位序图 .....	38
图 3-11 PAM4 模式下 Symbol 在 Lane 上的传输位序图 .....	38
图 3-12 AMCTL NRZ 模式下传输位序图 .....	39
图 3-13 AMCTL PAM4 模式下传输位序图 .....	39
图 3-14 NRZ 模式下 LMB 传输位序图 .....	40
图 3-15 数据链路层包第一个 Flit Symbol 示意图 .....	41
图 3-16 FEC Bypass 数据分发及成帧示意图 .....	41
图 3-17 RS(128,120)FEC 模式下 Flit Symbol 示意图 .....	42
图 3-18 RS(128,120)FEC 交织在 x8 链路上符号分发示意图 .....	42
图 3-19 RS(128,120) FEC 交织模式数据分发及成帧示意图 .....	43
图 3-20 RS(128,120)FEC 非交织分发示意图 .....	44
图 3-21 PAM4 电平示意图 .....	46
图 3-22 预编码示意图 .....	47
图 3-23 端口类型协商流程图 .....	58

## 图目录

图 3-24 QDLWS 链路宽度减少流程图 .....	60
图 3-25 QDLWS 链路宽度增加流程图 .....	61
图 3-26 Link_Active 下快速降 Lane 流程图 .....	63
图 3-27 FEC/CRC 模式切换协商流程图 .....	65
图 3-28 UB LMSM 状态跳转图 .....	66
图 4-1 数据链路层概览 .....	87
图 4-2 数据链路状态机 .....	88
图 4-3 DLLDP、DLLDB 和 Flit 关系图 .....	90
图 4-4 Padding 填充示例 .....	90
图 4-5 CRC 模式 DLLDP 格式 .....	91
图 4-6 DLLDP 第一个 Flit 格式 .....	92
图 4-7 PLENGTH 换算示例 DLLDP/DLLDB 格式 1 .....	94
图 4-8 PLENGTH 换算示例 DLLDP/DLLDB 格式 2 .....	94
图 4-9 PLENGTH 换算示例 DLLDP/DLLDB 格式 3 .....	95
图 4-10 Middle Block 和 Last Block 的第一个 Flit 格式 .....	95
图 4-11 CRC 模式 DLLDB 的最后一个 Flit 格式 .....	96
图 4-12 Non-CRC 模式 DLLDP 格式 .....	98
图 4-13 Non-CRC 模式 DLLDP 最后一个 Flit 格式 .....	98
图 4-14 CRC 模式 DLLCB 格式 .....	99
图 4-15 Non-CRC 模式 DLLCB 格式 .....	99
图 4-16 CRC 模式 Null Block 格式 .....	102
图 4-17 CRC 模式 NOB 格式 .....	103
图 4-18 CRC 模式 Retry_Idle Block 格式 .....	103
图 4-19 CRC 模式 Retry_Req Block 格式 .....	104
图 4-20 CRC 模式 Retry_Ack Block 格式 .....	104
图 4-21 CRC 模式 Crd_Ack Block Flit0 格式 .....	105
图 4-22 CRC 模式 Crd_Ack Block Flit1 格式 .....	105
图 4-23 CRC 模式 Param_Exchg Block 格式 Flit0 .....	106

## 图目录

图 4-24 CRC 模式 Param_Exchg Block 格式 Flit1~Flit(N-2).....	107
图 4-25 CRC 模式 Param_Exchg Block 格式 Flit(N-1).....	107
图 4-26 CRC 模式 Lane_Manage Block 格式 .....	108
图 4-27 CRC 模式 Block_Mode_Chg Block 格式.....	110
图 4-28 封装模式切换流程 .....	111
图 4-29 CRC 模式 Init Block Flit0 格式.....	112
图 4-30 CRC 模式 Init Block Flit1 格式.....	112
图 4-31 CRC 模式 Init Block Flit2 格式.....	112
图 4-32 CRC 模式 Init Block Flit3 格式.....	113
图 4-33 CRC 模式 Init Block Flit4~Flit(N-2)格式 .....	113
图 4-34 CRC 模式 Init Block Flit(N-1)格式 .....	113
图 4-35 CRC 模式定界流程 .....	116
图 4-36 Non-CRC 模式定界流程 .....	118
图 4-37 LLDP 发送流程 ( CRC 模式 ) .....	119
图 4-38 信用流控过程 .....	124
图 4-39 CRC 计算位序 .....	126
图 4-40 Retry Buffer 空间管理示意图 .....	128
图 4-41 重传请求状态机 .....	129
图 4-42 重传应答状态机 .....	131
图 4-43 正常重传处理流程示意图 .....	132
图 4-44 Retry_Req_Set 超时重传流程示意图 .....	133
图 5-1 网络层概览 .....	136
图 5-2 基于 16-bit CNA 地址格式的网络层包头 .....	137
图 5-3 基于 24-bit CNA 地址格式的网络层包头 .....	138
图 5-4 基于 IP 地址格式的网络层包头 .....	138
图 5-5 网络地址格式 .....	140
图 5-6 网络层路由机制 .....	141
图 5-7 UB Controller 的路由机制示意图 .....	142

图 5-8 NPI 定义 .....	142
图 5-9 UB Controller 工作模式 .....	143
图 5-10 UB Switch 的网络隔离 .....	144
图 5-11 CAQM 拥塞标记模式 .....	144
图 5-12 FECN 拥塞标记模式 .....	145
图 5-13 FECN_RTT 拥塞标记模式 .....	145
图 5-14 典型死锁场景示意图 .....	146
图 5-15 bit 翻转 ICRC 计算示例 .....	148
图 6-1 传输层概览 .....	149
图 6-2 TP Packet 格式 .....	150
图 6-3 TPACK/TPNAK 格式 .....	151
图 6-4 TPACK-CC/TPNAK-CC 格式 .....	151
图 6-5 TPSACK 格式 .....	151
图 6-6 TPSACK-CC 格式 .....	151
图 6-7 CNP 格式 .....	151
图 6-8 RTPH 格式 .....	151
图 6-9 UTPH 格式 .....	152
图 6-10 CTPH 格式 .....	152
图 6-11 LDCP CETPH 格式图 .....	156
图 6-12 CAQM CETPH 格式 .....	156
图 6-13 CNP CETPH 格式 .....	157
图 6-14 SAETPH 格式 .....	158
图 6-15 可靠传输服务流程 .....	159
图 6-16 PSN 区间图 .....	161
图 6-17 PSN 使用示例 .....	162
图 6-18 GoBackN 重传-配合快速重传，首次 TP Packet 丢失传输流程 .....	166
图 6-19 GoBackN 重传-配合快速重传，非首次 TP Packet 丢失传输流程 .....	167
图 6-20 GoBackN 重传-配合快速重传，尾 TP Packet 丢失传输流程 .....	168

## 图目录

图 6-21 GoBackN 重传-配合快速重传, TPACK 丢失传输流程 .....	169
图 6-22 GoBackN 重传-配合快速重传, 尾 TPACK 丢失传输流程 .....	169
图 6-23 GoBackN 重传-配合快速重传, TPNAK 丢失传输流程 .....	170
图 6-24 GoBackN 重传-无快速重传, 首次检测到 TP Packet 乱序 (丢包未实际发生) 传输流程 .....	172
图 6-25 GoBackN 重传-无快速重传, 首次检测到 TP Packet 乱序 (丢包实际发生) 传输流程 .....	173
图 6-26 选择性重传-配合快速重传, 首次 TP Packet 丢失传输流程 .....	177
图 6-27 选择性重传-配合快速重传, 非首次 TP Packet (无 MarkPSN 机制) 丢失传输流程 .....	178
图 6-28 选择性重传-配合快速重传, 非首次 TP Packet (使用 MarkPSN 机制) 丢失传输流程 .....	179
图 6-29 选择性重传-配合快速重传, 尾 TP Packet 丢失传输流程 .....	180
图 6-30 选择性重传-配合快速重传, TPSACK 丢失传输流程 .....	181
图 6-31 选择性重传-无快速重传, 首次 TP Packet 丢失 (丢包未实际发生) 传输流程 .....	182
图 6-32 选择性重传-无快速重传, 首次 TP Packet 丢失 (丢包实际发生) 传输流程 .....	183
图 6-33 TPG 与事务层交互示意图 .....	185
图 6-34 Write 操作流程图 (ROI 和 ROT 模式) .....	195
图 6-35 Write 操作流程图 (ROL 模式) .....	196
图 7-1 事务层概览 .....	197
图 7-2 完整 BTAH 格式 .....	198
图 7-3 精简 BTAH 格式 .....	198
图 7-4 完整 ATAH 格式 .....	203
图 7-5 精简 ATAH 格式 .....	203
图 7-6 完整 MAETAH 格式 .....	206
图 7-7 精简 MAETAH 格式 .....	207
图 7-8 MTETAH 格式 .....	208
图 7-9 TVETAH 格式 .....	209
图 7-10 TAIDETAH 格式 .....	209
图 7-11 OFSTETAH 格式 .....	210
图 7-12 IMMETAH 格式 .....	210
图 7-13 NTFETAH 格式 .....	211

## 图目录

图 7-14 BEETAH 格式 (以 8B 为例) .....	212
图 7-15 UDETAH 格式.....	212
图 7-16 可靠事务服务交互流程 .....	213
图 7-17 可靠事务服务重传流程 .....	214
图 7-18 不可靠事务服务交互流程.....	215
图 7-19 ROI 模式交互流程 .....	218
图 7-20 ROT 模式交互流程 .....	219
图 7-21 ROT 模式动态申请资源.....	219
图 7-22 ROL 模式交互流程 .....	221
图 7-23 Write 事务处理流程 .....	223
图 7-24 Write_with_notify 事务处理流程 ( ROI 模式 ) .....	225
图 7-25 Write_with_notify 事务处理流程 ( ROT 模式 ) .....	226
图 7-26 Write_with_notify 事务处理流程 ( ROL 模式 ) .....	226
图 7-27 Read 事务处理流程 .....	228
图 7-28 Atomic 事务处理流程.....	230
图 7-29 Send 事务处理流程 .....	234
图 7-30 Send_with_immediate 事务处理流程 .....	236
图 7-31 Prefetch_tgt 事务处理流程 .....	238
图 7-32 Management 事务处理流程.....	239
图 8-1 Jetty Group 模型 .....	242
图 8-2 Jetty 多对多通信模型 .....	243
图 8-3 Jetty 一对一对通信模型 .....	244
图 8-4 Jetty 状态机.....	244
图 8-5 公知 Jetty 通信交互流程 .....	247
图 8-6 TCP/IP 通信交互流程 .....	247
图 8-7 内存动态申请释放.....	248
图 8-8 内存池化使用方式.....	248
图 8-9 Ownership 状态转移图 .....	249

## 图目录

图 8-10 内存操作死锁避免 .....	251
图 8-11 消息操作死锁避免.....	252
图 8-12 URPC 职能角色.....	254
图 8-13 URPC 平等协议架构 .....	255
图 8-14 URPC 参数传递方式 .....	256
图 9-1 Home-User 访问模型 .....	259
图 9-2 以 UBA 作为索引查找 MATT 和 MAPT .....	260
图 9-3 访问处理流程示意.....	261
图 9-4 配置查找示意.....	261
图 9-5 目标实例配置表表项 .....	262
图 9-6 线性 TCT 结构和查找流程示意.....	267
图 9-7 目标上下文表表项.....	268
图 9-8 两级 TCT 结构和查找流程示意 .....	272
图 9-9 Level 1 目标上下文表表项.....	272
图 9-10 虚拟化场景下 UMMU 整体功能流程示意.....	273
图 9-11 Stage 1 和 Stage 2 地址转换 .....	273
图 9-12 两阶段地址转换流程示意.....	274
图 9-13 仅使用 MAPT Base Block 时，各级 MAPT 索引方式示意 .....	275
图 9-14 仅使用 MAPT Base Block 时权限查找流程示意 .....	275
图 9-15 内存访问权限表 Block Table 表项 .....	276
图 9-16 使用多个 MAPT Block 时，各级 MAPT 索引方式示意 .....	276
图 9-17 使用多个 MAPT Block 时权限查找流程示意 .....	277
图 9-18 单表项 MAPT 权限查找流程示意 .....	277
图 9-19 唯一内存访问权限表表项 .....	277
图 9-20 4KB 粒度下的多级 MAPT 权限查找流程示意 .....	280
图 9-21 Level 0 内存访问权限表表项 .....	280
图 9-22 Level 1 内存访问权限表表项 .....	280
图 9-23 4KB 粒度下的 Level 2 内存访问权限表表项 .....	281

## 图目录

图 9-24 2MB 粒度下的 Level 2 内存访问权限表表项.....	281
图 9-25 Level 3 内存访问权限表表项 .....	281
图 9-26 TokenValue 校验.....	284
图 9-27 L0 Page Table 查找结果为 L0 PTE 时, UB Decoder 整体转换流程.....	286
图 9-28 L0 Page Table 查找结果为 L0 PTRE 时, UB Decoder 转换流程.....	286
图 10-1 UB Domain 示意图 .....	288
图 10-2 配置管理模型结构示意图.....	290
图 10-3 Entity 配置空间与资源空间.....	290
图 10-4 资源管理模型结构示意图.....	291
图 10-5 EID 示意图 .....	291
图 10-6 Entity 间访问示意图.....	292
图 10-7 Entity 间访问隔离示意图 .....	293
图 10-8 基于 MMIO 的功能调用示意图.....	293
图 10-9 基于 Message 的功能调用示意图.....	294
图 10-10 中断机制示意图 .....	294
图 10-11 Type 2 寄存器结构示意图 .....	296
图 10-12 USI 消息 PDU 格式 .....	298
图 10-13 本地事件通知机制 .....	299
图 10-14 Error Message 事件上报流程示意图 .....	300
图 10-15 远端事件通知流程示意图 .....	301
图 10-16 配置空间访问机制示意图 .....	302
图 10-17 配置空间 Slice 结构示意图 .....	302
图 10-18 资源空间整体布局 .....	306
图 10-19 资源空间 0 .....	306
图 10-20 枚举管理消息格式 .....	307
图 10-21 配置管理消息格式 .....	307
图 10-22 Scan Header 格式 .....	308
图 10-23 Scan PDU 首 DW 格式.....	309

## 图目录

图 10-24 拓扑查询请求 Scan PDU 格式 .....	310
图 10-25 拓扑查询响应 Scan PDU 格式 .....	311
图 10-26 拓扑查询响应的 TLV 结构 .....	311
图 10-27 分片信息 .....	312
图 10-28 端口数量信息 .....	312
图 10-29 端口信息 .....	313
图 10-30 设备能力和配置信息 .....	313
图 10-31 配置 CNA 请求 Scan PDU 格式 .....	314
图 10-32 配置 CNA 响应 Scan PDU 格式 .....	315
图 10-33 查询 CNA 请求 Scan PDU 格式 .....	315
图 10-34 查询 CNA 响应 Scan PDU 格式 .....	316
图 10-35 Error Message PDU 格式 .....	318
图 10-36 Link Change Message PDU 格式 .....	319
图 10-37 Hot-plug Message PDU 格式 .....	319
图 10-38 配置请求消息 PDU 格式 .....	319
图 10-39 配置响应消息 PDU 格式 .....	320
图 10-40 VDM for UB Spec PDU 格式 .....	320
图 10-41 VDM for Vendor PDU 格式 .....	321
图 10-42 Resource Space Information Exchange 请求消息 PDU 格式 .....	321
图 10-43 Resource Space Information Exchange 响应消息 PDU 格式 .....	322
图 10-44 Entity 信息获取响应消息 PDU 格式 .....	322
图 10-45 链路邻居查询请求消息 PDU 格式 .....	323
图 10-46 链路邻居查询响应消息 PDU 格式 .....	323
图 10-47 Token 校验配置请求消息 PDU 格式 .....	324
图 10-48 Token 校验配置响应消息 PDU 格式 .....	324
图 10-49 Entity 注册请求 PDU 格式 .....	325
图 10-50 Entity 注销请求消息 PDU 格式 .....	326
图 10-51 Bus Instance 创建请求消息 PDU 格式 .....	327

## 图目录

图 10-52 Bus Instance 销毁请求消息 PDU 格式 .....	327
图 10-53 Configuration Completion Notification 请求消息 PDU 格式 .....	328
图 10-54 Port Reset Notification 请求消息 PDU 格式 .....	328
图 10-55 池化 Entity 管理示意图 .....	330
图 10-56 注册流程示意图 .....	330
图 10-57 注销流程示意图 .....	331
图 10-58 标准传输模式通信控制参考流程 .....	332
图 10-59 远端内存注册控制参考流程 .....	332
图 10-60 基于 UB 的虚拟化系统 .....	333
图 10-61 虚拟化机制 .....	334
图 10-62 错误记录 .....	342
图 11-1 设备接入认证 .....	346
图 11-2 用户粒度权限无效化 .....	350
图 11-3 集中式的 CIP 信道建立流程 .....	351
图 11-4 分布式的 CIP 信道建立流程 .....	352
图 11-5 TEE 扩展模型 .....	355
图 11-6 TEE 扩展过程 .....	358
图 11-7 UB 地址空间隔离 .....	359
图 11-8 TEE 扩展配置流程参考 .....	360

## 表目录

<b>表 3-1 UB 物理层模式 .....</b>	18
<b>表 3-2 FEC 编码器生成多项式系数表 <math>gi</math> (十进制) .....</b>	21
<b>表 3-3 RS FEC 码字列表.....</b>	22
<b>表 3-4 不同链路速率下 RX Lane 之间最大 skew 要求 .....</b>	25
<b>表 3-5 eBCH(16,5)码字集合 .....</b>	27
<b>表 3-6 eBCH-16 码字位序调整表 .....</b>	29
<b>表 3-7 AMCTL 域段结构表 .....</b>	30
<b>表 3-8 AMCTL.LID 编码表.....</b>	31
<b>表 3-9 AMCTL CTRL_TYPE 编码功能映射表 .....</b>	32
<b>表 3-10 AMCTL CTRL_DETAIL 编码功能映射表 .....</b>	32
<b>表 3-11 PAM4 编码和电平映射表 .....</b>	46
<b>表 3-12 DLTB 定义表.....</b>	48
<b>表 3-13 CLTB 定义表.....</b>	51
<b>表 3-14 RLTB 定义表.....</b>	52
<b>表 3-15 ELTB 定义表.....</b>	55
<b>表 3-16 2.578125/4.0 Gbps 的 EEIB 序列 ( NRZ ) .....</b>	57
<b>表 3-17 25.78125 Gbps 的 EEIB 序列 ( NRZ ) .....</b>	57
<b>表 3-18 53.125 Gbps 的 EEIB 序列 ( PAM4 ) .....</b>	57
<b>表 3-19 106.25 Gbps 的 EEIB 序列 ( PAM4 ) .....</b>	57
<b>表 3-20 PAM4 Lane 极性翻转示例 .....</b>	64
<b>表 3-21 EQ 模式协商表 .....</b>	72
<b>表 4-1 CRC 模式 LPH 域段 .....</b>	92
<b>表 4-2 CRC 模式 LBH 域段 .....</b>	95
<b>表 4-3 CRC 模式 BCRC 域段 .....</b>	97
<b>表 4-4 END 域段 .....</b>	98
<b>表 4-5 DLLCB 的 LCH 域段含义.....</b>	100

表 4-6 DLLCB 类型 .....	101
表 4-7 CMD 域段 .....	108
表 4-8 Init Block 域段信息 .....	114
表 4-9 数据链路层初始化自协商规则 .....	120
表 5-1 基于 16-bit CNA 地址格式的网络层包头域段 .....	137
表 5-2 基于 24-bit CNA 地址格式的网络层包头域段 .....	138
表 5-3 基于 IP 地址格式的网络层包头域段 .....	139
表 5-4 UB 协议栈使用 IPv4 包头的说明 .....	139
表 5-5 UB 协议栈使用 IPv6 包头的说明 .....	139
表 5-6 UB 协议栈使用 UDP 包头的说明 .....	140
表 5-7 RT 域段定义 .....	141
表 6-1 不同传输模式下支持的特性 .....	150
表 6-2 RTPH/UTPH/CTPH 域段 .....	152
表 6-3 LDCP CETPH 域段 .....	156
表 6-4 CAQM CETPH .....	156
表 6-5 CNP CETPH .....	157
表 6-6 SAETPH .....	158
表 6-7 重传算法推荐使用场景描述 .....	163
表 6-8 动态超时重传间隔表 .....	164
表 6-9 TPG 示例 .....	185
表 6-10 CC Context 中与拥塞控制相关的变量 .....	188
表 6-11 窗口调整规则 .....	190
表 7-1 BT AH 域段 .....	198
表 7-2 不同事务操作包格式 .....	202
表 7-3 AT AH 域段 .....	203
表 7-4 事务响应包格式 .....	206
表 7-5 MAET AH 域段 .....	207
表 7-6 MTET AH 域段 .....	208

## 表目录

表 7-7 TVETAH 域段 .....	209
表 7-8 TAIDETAH 域段 .....	209
表 7-9 OFSTETAH 域段 .....	210
表 7-10 IMMETAH 域段 .....	211
表 7-11 NTFETAH 域段 .....	211
表 7-12 BEETAH 域段 .....	212
表 7-13 事务执行序示例 .....	216
表 7-14 事务层和下层配合关系（针对有执行序要求的事务） .....	221
表 7-15 事务层和下层配合关系（针对无执行序要求的事务） .....	222
表 7-16 Atomic 事务子类型 .....	231
表 8-1 公知 Jetty 编号 .....	247
表 8-2 URPC 参数传递方式适用场景 .....	256
表 9-1 目标实例配置表表项域段描述 .....	262
表 9-2 目标上下文表表项域段描述 .....	268
表 9-3 L1 目标上下文表表项域段描述 .....	272
表 9-4 内存访问权限表 Block Table 表项域段描述 .....	276
表 9-5 唯一内存访问权限表表项域段描述 .....	278
表 9-6 多级 MAPT 下各级 MAPTE 域段描述 .....	281
表 9-7 TCTE 排他性校验规则 .....	285
表 9-8 MAPTE 排他性校验规则 .....	285
表 10-1 Type 1 中断配置寄存器组 .....	295
表 10-2 Type 2 中断能力寄存器组 .....	296
表 10-3 Type 2 中断地址表结构 .....	297
表 10-4 Type 2 中断向量表结构 .....	298
表 10-5 Slice Header 寄存器定义 .....	303
表 10-6 配置空间 Slice 的地址空间定义 .....	303
表 10-7 配置空间寄存器属性定义 .....	305
表 10-8 管理命令字 .....	310

## 表目录

<b>表 10-9 拓扑查询响应的 TLV 类型</b>	312
<b>表 10-10 MGMTETAH 格式</b>	316
<b>表 10-11 MGMTETAH Code and Sub Code 编码</b>	317
<b>表 10-12 A 类错误列表</b>	337
<b>表 10-13 B 类错误列表</b>	338
<b>表 10-14 C 类错误列表</b>	338
<b>表 11-1 安全功能与安全威胁的映射</b>	344
<b>表 11-2 访问控制功能应用场景</b>	347
<b>表 11-3 两种互联模式的区别</b>	357