SiFive TileLink 规格书
1.7.1 草案 预发布版本（中文）

2019-06-21

SiFive TileLink规格书（中文版）

版权声明

变更历史

版本	日期	修改
1.7.1-draft	2017年8月22日	预发布版

早期预览版注意事项

此早期预览版只是为了方便早期的审校，且因为其内容可能修改故可能并不适合用作具体的实现

中文翻译团队

- 翻译：刘鹏 林容威 李沛南
- 校对：宋威 郭雄飞

目录

第 1 章

介绍

TileLink 是一个芯片级(chip-scale)互连标准，允许多个主设备(masters)，以支持一致性的存储器映射(memory-mapped)方式访问存储器和其他从设备(slave)。TileLink 的设计目标，是为片上系统(System-on-Chip)提供一个具有低延迟和高吞吐率(high-throughput)传输的高速、可扩展的片上互连方式，来连接通用多处理器(multiprocessors)、协处理器、加速器、DMA以及各类简单或复杂的设备。总结来说，TileLink是：

- 免费开放的紧耦合、低延迟的 SoC 总线
- 为 RISC-V 设计，也支持其他 ISAs
- 提供物理寻址(physically addressed)、共享主存(shared-memory)的系统
- 可用于建立可扩展的、层次化结构的和点对点的网络
- 为任意数量的缓存或非缓存主设备提供一致性的访问
- 支持从单一简单外设到高吞吐量的复杂多外设的所有通讯需求

TileLink 还具备以下重要的特性：

- Cache 一致性的内存共享系统，支持兼容 MOESI 的一致性协议
- 对任何遵守该协议的SoC系统来说，可验证确保无死锁
- 使用乱序的并发操作以提高吞吐率
- 使用完全解耦的通讯接口，有利于插入寄存器来优化时序
- 总线宽度的透明自适应和突发传输序列的自动分割
- 针对功耗优化的信号译码

协议扩展级别

TileLink 支持多种类型的通讯代理模块，并定义了三个从简单到复杂的的协议扩展级别。每个级别定义了兼容该级别的通讯代理模块需要支持的协议扩展子集，如表 1.1 所示。最简单的是 TileLink 无缓存轻量级(Uncached Lightweight, TL-UL)，只支持简单的单个字读写(Get/Put)的存储器操作。相对复杂的 TileLink无缓存重量级(Uncached Heavyweight, TL-UH)，添加了预处理(hints)、原子访问和突发访问支持，但不支持一致性的缓存访问。最后，缓存相关级别 TileLink Cached(TL-C)是最复杂（完整）的协议，支持使用一致性的缓存模块。

	TL-UL	TL-UH	TL-C
缓存块传输			y

	TL-UL	TL-UH	TL-C
使用BCE通道			y
多包突发传输		y	y
原子访问		y	y
预处理(hint)访问		y	y
读/写(Get/Put)访问	y	y	y

表 1.1: TileLink 协议扩展级别

当一个处理器的TL-C通讯代理模块和一个外设的TL-UL通讯代理模块通信时，要么处理器代理避免使用更高级的特性，要么两者之间必须使用一个 TL-C 到 TL-UL 的适配器。TileLink中的各种代理可以支持一些其他特性的组合，但本规范只包括上述三种扩展级别。

文档概述

余下的规范分为以下几节：

- 第2章 给出 TileLink 架构和常见抽象的概述
- 第3章 定义每个 TileLink 通道所需的特定信号
- 第4章 定义这些信号如何被用来交换 TileLink 信息
- 第5章 概述 TileLink 代理的可用操作，提供有关字节序，地址空间的使用和事务标识符(transaction identifier)的指导
- 第6章 详解在 TileLink 上如何使用消息(messages)执行基本的读写操作
- 第7章 对 TileLink 进行扩展，支持突发传输，原子操作和预处理
- 第8章 概述完整的 TileLink 协议如何管理缓存块传输

第 2 章

架构

Tilelink 协议适用于在代理(agent)互联拓扑图中，完成消息(message)的传递。共享地址空间的代理经由点对点的通道(channel)收发消息，来完成操作(operation)，该通道被称为链路(link)。

- 操作(operation)：在特定地址范围内，改动存储数据的内容、权限或是其在多级缓存中的存储位置。
- 代理(agent)：在协议中，为完成操作负责接收、发送消息的有效参与者。
- 通道(channel)：一个用于在主(master)接口和从(slave)接口之间传递相同优先级消息的单向通信连接。
- 消息(message)：经由通道传输的控制和数据信息。
- 链路(link)：两个代理之间完成操作所需的通道组合。

网络拓扑

代理间通过链路(link)互相连接。每条链路的一端连接一个代理的主接口，另一端则连接另一个代理的从接口。主接口一端的代理可以向另一端代理发送请求，让其执行访存、获取权限以传输或缓存数据的操作。从接口端的代理是对应地址空间的访问和权限管理者，依据主接口发来的请求执行相应的访存操作。

图 2.1：最基本的 TileLink 网络操作。两个模块(Module)通过链路相连。左侧模块内的代理包含一个主接口，右侧模块内的代理包含一个从接口。带有主接口的代理向带有从接口的代理发起请求。如果需要，从接口端的代理可能和更底层的存储器进行通信。在获取到请求的数据或权限之后，从接口端的代理则通过消息响应原始请求。

图 2.1 TileLink 网络中链路示例。该链路使用两条通道连接一对主从接口。当主接口需要在共享的存储器上执行某个操作时，它通过请求通道向从接口发送一个请求消息，然后在响应通道上等待从接口的回复的确认消息。

TileLink 支持多种网络拓扑。具体来说，如果将代理抽象为图论中的节点，而将链路抽象为从主接口指向从接口的有向边，那么任何有能被描述为有向无环图的拓扑结构都可以被支持。图 2.2 展示了一个典型的拓扑结构。该拓扑中的交换器(Crossbar) 和缓存(Cache)模块都包含一个兼有主接口(右端)和从接口(左端)的代理节点。

图 2.2： 一个更复杂的 TileLink 网络拓扑。其中的两个模块都包含一个既有主接口和从接口的代理。

重要提示：单个硬件模块可包含多个独立的 TileLink 代理。如图 2.3 所示，交换器采用一个代理完成路由，管理来自输入链路的到输出链路的数据传输，采用另一个代理配置访

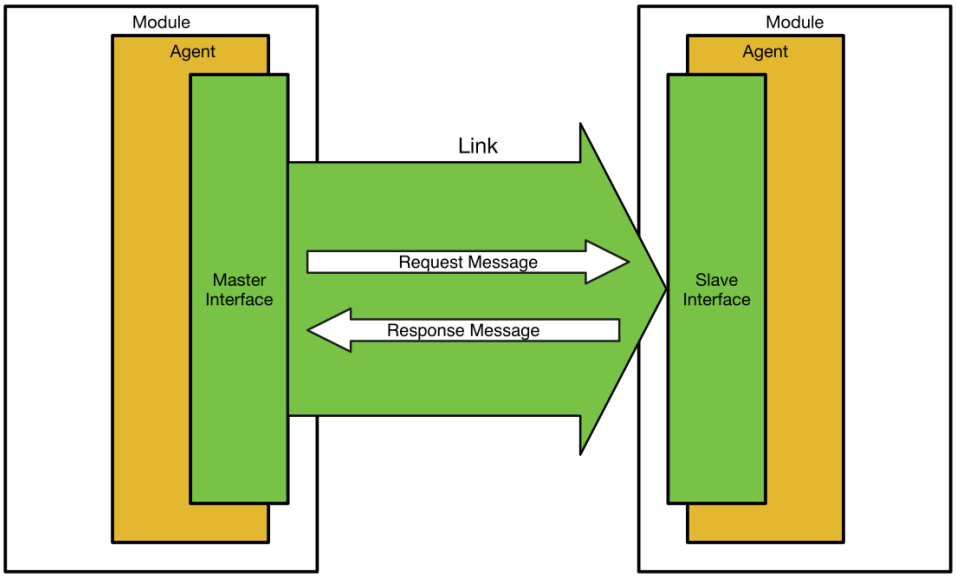

图 2.1: 最基本的 TileLink 操作

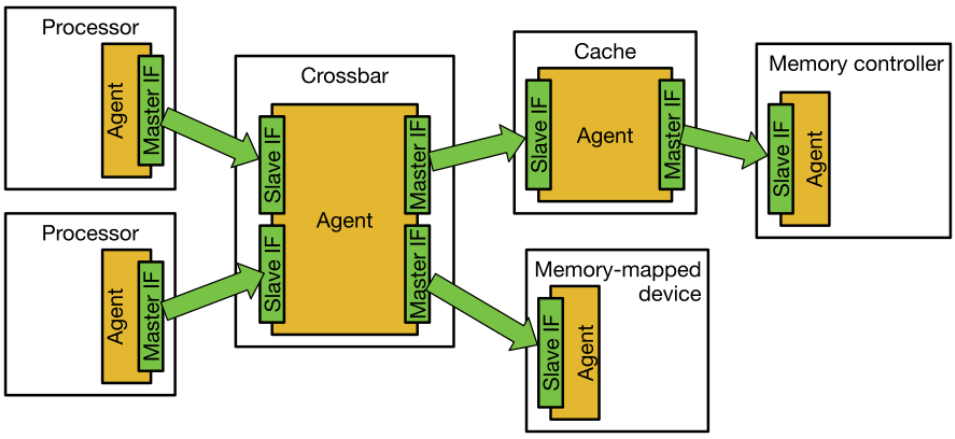

图 2.2: 一个更复杂的 TileLink 网络拓扑

问的状态。在协议内，对有向无环图(DAG)的定义仅适用于代理间的层次结构，而不是模块间。

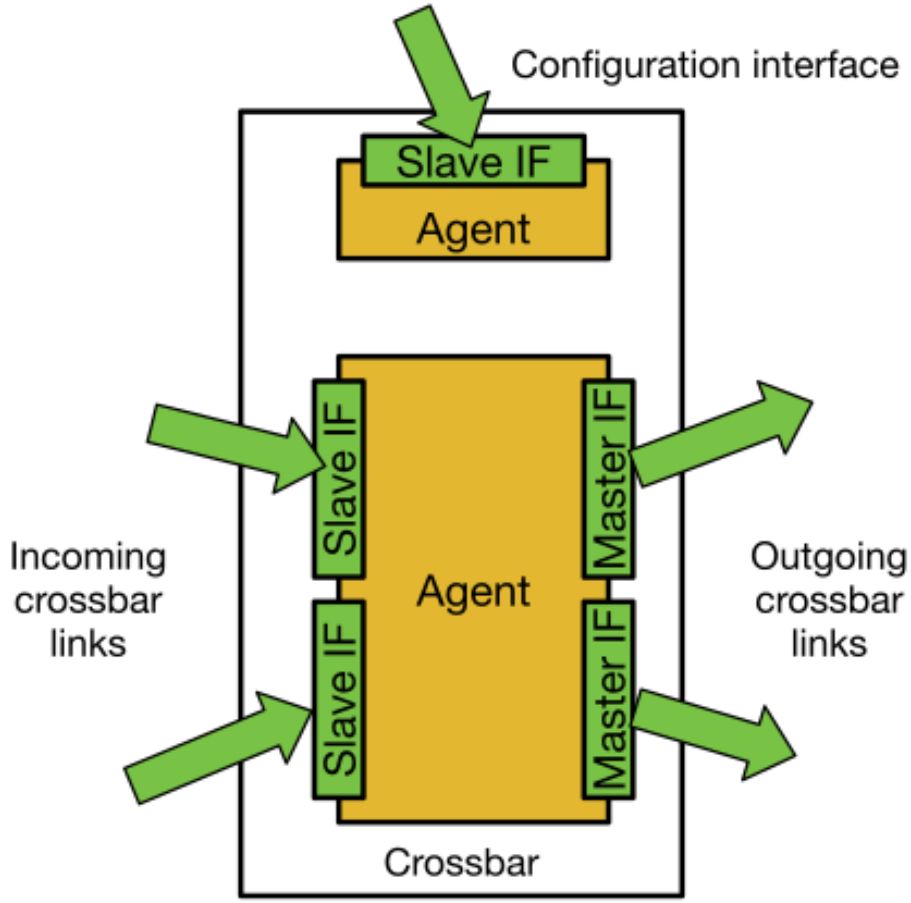

图 2.3: 包含两个代理的 crossbar 模块。

图 2.3： 包含两个代理的交换器模块。 下侧代理有多个主和从接口，用于在规划一般操作数据传输的路线，而上侧代理只有一个从接口，用于访问配置交换器的传输模式。

通道优先级

每个网络链路中，TileLink 协议定义了五个逻辑上相互独立的通道，代理可通过它们传递消息。为了避免死锁，TileLink 强制规定了这五个通道上传输消息的优先级。大部分通道包含控制传输的信号以及实际数据的副本。通道上消息的传递是单向的，主向从接口传送消息，或从向主接口传送消息。图 2.4 标明了五个通道的方向。

任何访存操作都需要两个最基本的通道：

通道 A: 传送一个请求，访问指定的地址范围或对数据进行缓存操作。

通道 D: 向最初的请求者传送一个数据回复响应或是应答消息。

最高协议兼容层 TL-C 额外包含另外三个通道，具备管理数据缓存块权限的能力：

通道 B: 传输一个请求，对主代理已缓存的某个地址上的数据进行访问或是写回操作。

通道 C: 响应通道 B 的请求，传送一个数据或是应答消息。

通道 E: 传输来自最初请求者的缓存块传输的最终应答，用于序列化。

图 2.4：代理间 TileLink 链路的五个通道。

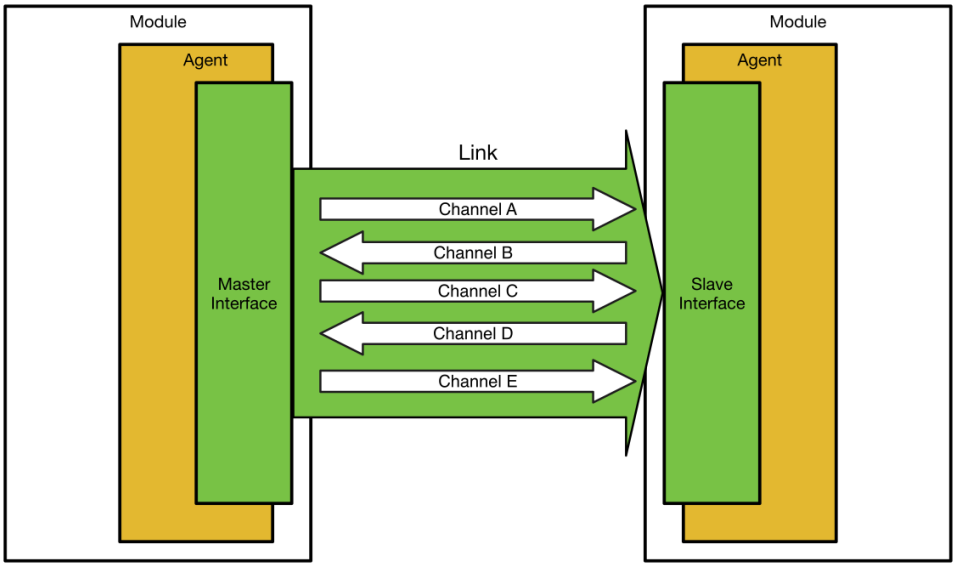

图 2.4: 在任意一对代理之间构成一条 TileLink 链路的五个通道。

各个通道传递消息的优先级顺序是 A << B << C << D << E 。设置优先级保证了消息在 TileLink 网络的传输过程中不会进入路由环路或是资源死锁。换句话说，代理间消息在所有通道上的传输过程仍然保持为有向无环图。这对 TileLink 保证无死锁来说是一个必要的特性。

地址空间属性

操作目标地址段的属性限制了哪些类型的消息才可以在一个 TileLink 网络中传播。一段地址空间的属性包括：TileLink 兼容层(conformance level)，存储一致性模型，是否可缓存(cacheability)，FIFO 顺序要求(ordering requirements)，是否可执行(executeability)， 特权层(privelege)以及服务质量保证(Quality-of-Service guarantees)。

依据这些属性，TileLink 将特定地址段可以执行何种操作与消息内容本身分开。在代理发送最初的请求消息之前，就决定一个操作是否合法，使得 TileLink 可以避免许多错误。

协议并不涉及哪些地址范围应具备哪些属性，以及这些属性反过来如何定义合法消息。

第 3 章

信号描述

这章用表格的形式介绍了 TileLink 五个通道使用到的所有信号，汇总在表 3.3。当结合每个通道的方向，表 3.2 的信号类型决定了信号的方向。这些信号的宽度由表 3.3 描述的值进行参数化。

通道	方向	目的
A	主端到从端	发送到一个地址的请求消息
B	从端到主端	发送到一个缓存块的请求消息 (TL-C独有)
C	主端到从端	从一个缓存块发回的响应消息(TL-C独有)
D	从端到主端	从一个地址发回的响应消息
E	主端到从端	针对缓存块传输的最终握手消息(TL-C独有)

表 3.1: TileLink 通道汇总

类型	方向	描述
X	输入	时钟或者复位信号，作为两边 TileLink 代理的输入
C	通道顺向	控制信号，在簇发传输期间不会改变
D	通道顺向	数据信号，在每个传输周期(beat)都会改变
F	通道顺向	终结信号，只在最后的传输周期会改变
V	通道顺向	有效信号， 表明 C/D/F 包含有效数据
R	通道反向	就绪信号，表明已收到有效信号（ V ）

表 3.2: TileLink 信号类型。 通道方向如表 所示。

参数	描述
w	以字节为单位的数据总线宽度，必须是2的整次幂
a	地址的位宽，最小32位
z	大小(size)字段的位宽，最小4位
o	区分源（主）端所需的比特数
i	区分终（从）端所需的比特数

表 3.3: TileLink 每条链路参数

信号命名惯例

除了 clock 和 reset 信号之外，TileLink 信号名字的组成：通道识别符（a到e）跟着一个下划线，再接一个信号名（在下面小节中一一列举）。

对于拥有多个 TileLink 接口的设备，推荐在所有 TileLink 信号名字之前加上一些描述性的标示加下划线。例如，a_opcode 变成 gpio_a_opcode。

时钟、复位和电源

TileLink 是一个同步总线协议。在 TileLink 链路上的主接口和从接口都必须共享相同的时钟，复位和电源。然而，在拓扑结构里的不同链路可以是不同的时钟，复位和电源信号。

信号	类型	宽度	描述
clock	X	1	总线时钟，在时钟上升沿时对输入进行采样
reset	X	1	总线复位，高电平有效，可以异步置1，但必须跟时钟上升沿同步时才可以置0

表 3.4: TileLink 所有通道共有的时钟和复位信号

时钟

每个通道在时钟的上升沿对信号进行采样。输出信号只有在时钟的上升沿之后才可发生改变。

复位

在将复位信号置0之前, 主端必须把 a_valid, c_valid 和 e_valid 这些信号置为低电平，而从端必须把 b_valid 和 d_valid 必须置为低电平。在复位信号低电平、时钟信号第一个上升沿之后，可以把有效信号（ valid）置为高电平。 在复位信号置 1 时，有效信号（ valid）必须保持低电平至少 100 个时钟周期。

在复位期间，就绪、控制和数据信号可以取任意值。

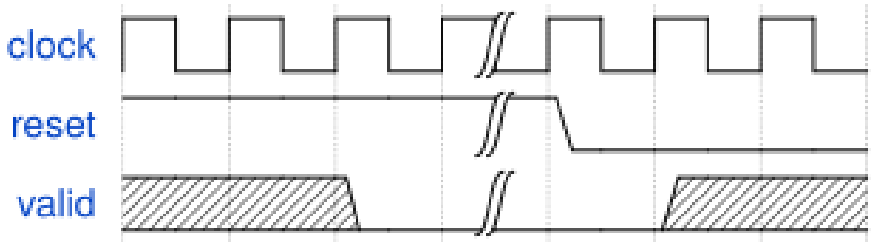

图 3.1: 有效信号（Valid）必须在复位期间至少 100 个周期内保持低电平

电源或跨时钟（Clock Crossing）

禁止在当 TileLink 一边还是上电状态时，对另一边进行下电操作。

如果 TileLink 必须跨电源或是时钟域，需要一个 TileLink 到 TileLink 的适配器，在一个域用作为从接口而在另一个域用作主接口。这个适配器的两个接口各自上下电，使用不同的时钟、复位信号而不相互造成影响。

我们应该让这样的跨越行为小心地跟 SoC 的其他部分协调一致，确保要跨越的一边复位或是下电时没有尚未处理的 TileLink 请求。如果 TileLink 消息一旦丢失或是重复，将会导致整个 TileLink 总线死锁。

通道 A（强制）

通道 A 的传输方向是从主接口到从接口，携带请求消息发送到一个特定的地址。所有 TileLink 兼容层都要强制使用这个通道。

信号	类型	宽度	描述
a_opcode	C	3	操作码。识别该通道携带消息的类型。（表 5.2）
a_param	C	3	参数码。具体意义视 a_opcode 而定，可用来表明缓存操作许可的传递或者字操作码。(6.2, 7.2, 8.3节)
a_size	C	z	操作大小的对数：2^n 个字节。（4.6 节）
a_source	C	o	唯一的，每条链路 master 来源（source）的 ID。（5.4 节）
a_address	C	a	操作的按字节寻址的地址目标。必须跟 a_size 对齐。（4.6 节）
a_mask	D	w	选择消息数据的哪个字节（Byte lane）。（4.6 节）
a_data	D	8w	可随消息携带的数据。（4.6 节）
a_valid	V	1	通道携带的数据是否有效。（4.1 节）
a_ready	R	1	通道数据是否已被接收（4.1 节）。

表 3.5: 通道 A 信号

通道 B（TL-C独有）

通道 B 的传输方向是从从接口到主接口，用于向保存一个特定缓存块的主代理发送请求消息。该通道被用于 TL-C 兼容层（conformance level）而对于较低层是可选的。

信号	类型	宽度	描述
b_opcode	C	3	操作码。识别该通道携带消息的类型。（表 5.2）
b_param	C	3	参数码。具体意义视 b_opcode 而定，可用来表明缓存操作许可的传递或者字操作码。(8.3节)
b_size	C	z	操作大小的对数：2^n 个字节。（4.6 节）
b_source	C	o	唯一的，每条链路 master 来源（source）的 ID。（5.4 节）
b_address	C	a	操作的按字节寻址的地址目标。必须跟 b_size 对齐。（4.6 节）
b_mask	D	w	选择消息数据的哪个字节（Byte lane）。（4.6 节）
b_data	D	8w	可随消息携带的数据。 (4.6 节)
b_valid	V	1	通道携带的数据是否有效。（4.1 节）
b_ready	R	1	通道数据是否已被接收。（4.1 节）

表 3.6: 通道 B 信号

通道 C（TL-C独有）

通道 C 的传输方向是从主接口到从接口。携带对通道 B 请求作为响应的消息发送给一个特定缓存数据块。也被用于自发地（voluntarily）写回脏缓存数据（dirtied cached data）。 该通道被用于 TL-C 兼容层（conformance level）而对于较低层是可选的。

信号	类型	宽度	描述
c_opcode	C	3	操作码。识别该通道携带消息的类型。（表 5.2）
c_param	C	3	参数码。具体意义视 c_opcode 而定，可用来表明缓存操作许可的传递或者字操作码。(8.3节)
c_size	C	z	操作大小的对数：2^n 个字节。（4.6 节）
c_source	C	o	唯一的，每条链路 master 来源（source）的 ID。（5.4 节）
c_address	C	a	操作的按字节寻址的地址目标。必须跟 c_size 对齐。（4.6 节）
c_data	D	8w	可随消息携带的数据。 (4.6 节)
c_error	F	1	主代理不能服务该请求。（4.5 节）
c_valid	V	1	通道携带的数据是否有效。（4.1 节）
c_ready	R	1	通道数据是否已被接收。（4.1 节）

表 3.7: 通道 C 信号

通道 D（强制的）

通道 D 的传输方向是从从接口到主接口。它携带对通道 A 发送到特定地址请求作出响应的消息。它还携带了对于通道 C 自发写回（voluntary writebacks）的应答。该通道被用于所有 TileLink 兼容层并且是必选的。

信号	类型	宽度	描述
d_opcode	C	3	操作码。识别该通道携带消息的类型。（表 5.2）
d_param	C	2	参数码。具体意义视 a_opcode 而定，可用来表明缓存操作许可的传递或者字操作码。(6.2, 7.2, 8.3节)
d_size	C	z	操作大小的对数：2^n 个字节。（4.6 节）
d_source	C	o	唯一的，每条链路 master 来源（source）的 ID。（5.4 节）
d_sink	C	i	唯一，每个链路 slave sink 的 ID（slave sink identifier）（5.4 节）
d_data	D	8w	可随消息携带的数据。(4.6 节)
d_error	F	1	从代理不能服务该请求。（4.5 节）
d_valid	V	1	通道携带的数据是否有效。（4.1 节）
d_ready	R	1	通道数据是否已被接收。（4.1 节）

表 3.8: 通道 D 信号

通道 E（TL-C独有）

通道 E 的传输方向是从主接口到从接口。它携带是否收到通道 D 响应消息的应答信号，这被用作操作序列化。该通道被用于 TL-C 兼容层（conformance level）而对于较低层是可选的。

信号	类型	宽度	描述
e_sink	C	i	唯一，每个链路 slave sink 的 ID（slave sink identifier）（5.4 节）
e_valid	V	1	通道携带的数据是否有效。（4.1 节）
e_ready	R	1	通道数据是否已被接收。（4.1 节）

表 3.9: 通道 E 信号

第 4 章

序列化

TileLink中的五个通道实现为五个物理隔离的单向并行总线。每个信道都有一个发送者和一个接收者。对于A、C和E通道，具有主接口的代理是发送者，具有从接口的代理是接收者。对于B通道和D通道，具有从接口的代理是发送者，具有主接口的代理是接收者。

许多TileLink消息包含有效的数据负载，而根据消息和数据总线的大小，可能需要跨多个时钟周期(或beats)发送。多拍的信息则通常被称为簇发(burst)。没有数据载荷的TileLink消息总是在单拍中完成。TileLink禁止在一个通道中插入来自不同消息的任何数据。一旦一个簇发开始，发送方在簇发的最后一拍被接收方接收之前，都不得发送任何来自其他消息的任一拍数据。簇发的持续时间由通道的size字段决定。

为了规范TileLink信道中的各拍数据流，接收者拉高通道信号以指示他们接收一拍数据的能力。接收者拉低信号，表示他们正忙，不能接收节拍。相反地，一拍数据的发送者拉高信道内的valid信号，以表明信道内有一拍数据。只有当ready和valid同时被拉高时，各拍数据才被有效接收成功。

本章的其余部分列出了流程控制和死锁避免规则，这些规则用于管理ready和valid相互交织的时候的情况。我们也定义了TileLink代理如何互相连接在一起，并定义了请求/响应消息对排序的规则。最后，我们将讨论与传统总线标准的接口、错误处理以及如何将簇发数据映射到特定宽度的物理数据总线上。

流程控制规则

为了实现正确的ready和valid信号握手，需要遵守以下规则：

- 当ready信号为低时，接收者不能处理数据并且发送者认为该拍内的数据未被处理。
- 当valid信号为低时，接收者应当认为该数据信号并不是一个正确的TileLink数据。
- valid必须与ready信号独立，如果一个发送者希望发送一拍数据，则必须使能一个valid信号，同时该valid信号是与接收者的ready信号独立的。
- 从ready到valid信号或者任何数据或控制信号之间不能存在任何组合逻辑路径。

一个接收者可以根据4.2小节的无死锁规则只保持ready为低

任何不被禁止的事情都是允许的。特别是，接收者可以驱动ready信号以作为valid或任何控制和数据信号的响应信号。例如，如果对一个繁忙的地址发出了valid请求，仲裁者可能会拉低ready。然而，只要可能，我们就建议独立地驱动ready，以减少握手电路的深度。

注意，发送者可能会拉高valid，然后在下一个周期拉低它，即使消息在上一个循环中没有被接收。例如，发送者可能有其他更高优先级的任务要在下一个周期中执行，而不是试图再次发送被拒绝的消息。此外，发送者可以在一个消息没有被接收时更改控制信号和数据信号的内容。

对于能够携带簇发的TileLink的通道，有着额外的规则约束。一个簇发一旦第一拍数据被接收，直到最后一拍数据传输结束都认为处于传输过程中。当一个数据包处于传输过程中时，如果valid信号为高，发送者必须满足：

- 只有同消息的簇发的一拍数据。
- 控制信号与第一拍相同。
- 数据为与之前拍数据的地址再加上按数据总线的字节宽度得到的地址所对应的数据。
- 只在最后一拍数据出现时改变的终结信号。

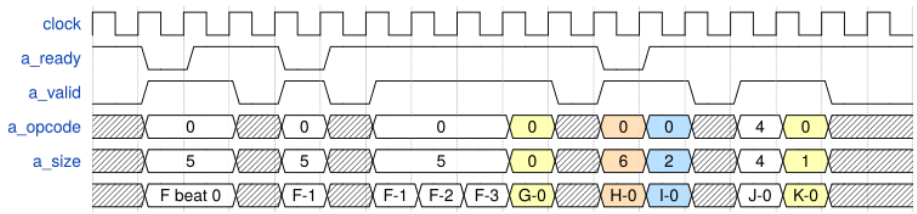

图 4.1: 在一个64比特的A通道中，6个消息的Ready-Valid信号实例

一个遵循这些规则的八个字节宽度的通道的波形图如图4.1所示。注意到所有控制和数据信号的有效性都是由valid拉高而预示的。只有在valid和ready的信号同时为高时数据才会被有效接收。

图中有六个信号，分别为F,G,H,I,J,K。其中a_size标识了信号的长度，如F的数据大小(size)为5，即为 2^5= 32字节，opcode标识了通道正进行的操作，0为PutFullData，4为Get操作。 图中有6个发送的消息：F,G,H,I,J,k。第一个消息F,大小为5,表明操作访问2^5 = 32个字节。操作码0是一个PutFullData消息，因此F携带有数据。因为通道A一拍可以传输携带8字节的数据，所以有4个节拍的数据需要交换。不同拍的数据分别表示为F-0 F-1 F-2和F-3。F-0出现的第一个周期，从端并不接收。主端选择重复F-0，随后该消息F-0被接收。在F-0被接收后，我们认为簇发F处于传输过程中。因此，主端别无选择，只能重复F-1直到被接收。然而，主端仍然可以在簇发中拉低valid。然后主端继续按照一定的顺序呈现F的不同拍数据，直到最后一拍F-3被接收为止。

第二个消息G的大小为0，表示1字节的消息。这适合于传输单拍数据，并立即交换。消息H(8拍簇发)是由主端发送的，但是被拒绝接收。由于第一拍的簇发没有被接收，簇发不算进入处理过程中，主端选择在接下来的周期中呈现一个不同的信息I。而消息H则不再需要发送了。

消息J操作码为4，在通道A上表示为一个Get消息。尽管如a_size所示，Get操作字节为16个字节，但消息J本身不携带数据，因此单个节拍即被接收。消息K则可以在下一个周期发送并被接收。

无死锁

TileLink是被设计为可以完全避免死锁的。为了保证一个TileLink拓扑网络永远不会死锁，我们提出两个一致性系统必须遵守的规则。首先，我们定义一个规则用于管理接收端代

理通过拉低ready来拒绝接收当前拍的信号，其次，我们定义一个有关TileLink中所允许的互联拓扑结构的规则。

通过将这两种规则与拓扑网络中通道的严格的优先级的规范结合，我们可以保证正确实现的TileLink不会发生死锁的情况。

规则中使用的定义

所有的TileLink操作组成了在不同代理之间传输的有序的消息序列。我们使用如下的名词来定义与死锁相关的规则，这个规则基于消息在整体顺序中的位置。

- 请求(request)消息：用于指定一个进行访存或权限转换的消息。
- 后续(follow-up)消息：表示接收某消息后需发送的消息。
- 响应(response)消息：一个与请求消息配对的必要的后续消息。
- 寄生(recursive)消息：任何在请求和响应消息之间嵌入的后续消息。
- 转发(forwarded)消息：也是一个寄生消息，和触发该消息的消息具有相同的优先级。

每个请求消息最终都必须得到一个响应消息的回复。一个响应消息总是有着比触发该消息的请求消息具有更高的优先级。一个消息可能既是请求又是响应，这样的消息会触发新的响应。

一个在请求W和响应Z中嵌入的寄生消息X 必然有着等于或高于W的权限，但比Z的权限低。X本身必须在W之后Z之前发送。如果寄生请求X消息有回复 Y，Y必须拥有等于或低于Z的权限，并且在X之后Z之前被接收。

一个有着与触发其的消息的相同优先级的寄生消息被称为转发消息。一个具有多主端接口及多从端接口的代理，并且能将消息从一个接口转发到其他接口，这样的代理叫做 转发代理。转发代理对于构建存储与总线层级结构拓扑图非常重要，并且他们的主端从端接口的ready信号如何地耦合也有相应的管理规则，如下所述。

代理的转发规则

接收者并不需要在valid信号为低时，将ready信号置为高。然而，当发送者的valid信号为高时，接收者的ready信号必须置为高，除非接收者有拒绝该拍的合理的理由。在TileLink中，只有四种情况，接收者可以通过拉低ready信号来拒绝一拍有效信号。

1.接收者可能选择进入一个有限的忙碌状态，在此期间，不会将ready信号拉高。

- 忙碌状态的持续时间不会超过某个固定周期数。
- 接收者可以随意进入一个忙状态，但是在忙状态期间，必须接收至少一拍数据。
- 例如，在周期性忙碌时期(例如一个DDR刷新操作)，这样的要求可以通过在控制器的前方放置一个深度为一的缓冲器实现。缓冲器会拉高ready直到缓冲器被填充。当控制器完成了刷新，它会清理缓冲器，并且处理被存储的消息，同时在一定周期内让缓冲器再次拉高ready信号。

2.当一条对请求消息响应消息在X通道上被拒绝，发送该响应的代理可以将所有优先级低于X的通道的ready信号无限期地拉低。

- 所有会触发该规则的回复消息如表5.3所示。

- 例如，一个简单从端在通道A接收了一个GET消息，并试图将回复消息AccessAckData从通道D中发送。如果该回复消息因为d_ready为低而被阻塞，那么从端可能会保持a_ready为低。
- 如果TL-C级别的从端代理在通道A接收了一个请求消息，并且在试图在通道D发出一个响应消息时被阻塞，本条规则不允许阻塞通道B、C，只能阻塞通道A。

3.当一条由通道X上的请求消息引发的寄生消息被拒绝时，发送端代理可能会无限期地拉低所有优先级低于X的通道的ready信号。(只与转发或TL-C代理有关)

- 例如，一个交换器试图在从端输出通道A前递一个Get信号，当交换器等待该消息被接收时，它可能会在所有主端通道A输入处保持ready信号为低。

4.当一条回复由通道X发送的响应消息没有被接收时，接收者可能会无限期地拉低所有优先级低于X的通道的ready信号。(只与转发或TL-C代理有关)

- 最后一拍数据已经被发送后，代理在等待该请求的回复消息。
- 例如，一个过去已经在从端通道A输出处转发过一个Get信号的交换器，在等待回复时，会将其所有主端通道A输入处的ready信号保持为低。

以上的这些规则非常详尽，在设计一个代理时，设计者必须确保以上的情况一旦发生，即使valid信号为高，也要使ready信号为低。一个不遵守以上规则的代理是不兼容于协议的，并且在整个TileLink网络中会影响转发的处理。

网络的拓扑规则

每一个TileLink网络都能被描述为一个代理图，并且该图能被用于判断该网络是否能够保证避免死锁。该图中，每一个代理都有节点，每条TileLink的链路被视为有向边。在代理图中，链路从主端指向从端。图4.2描述了一个TileLink代理图。蓝色框内代表了RTL级模块，黄色圆圈代表了代理。一个合法的TileLink系统必须是一个有向无环图。任何在图中的环都可能导致死锁。

为了说明限制，我们对TileLink代理需要更为准确的定义。一个直观上的定义是，当一条来自某个链路的消息，在没有首先通过任何其他TileLink链路情况下，触发了另一链路后续的消息时，我们认为这两条链路连接在同一个TileLink代理上。即使是两条链路连接在同一个RTL模块上，这并不代表该模块就是一个带有两条链路的代理，在这个模块内部可能有着两个不同的代理。

例如，一个连接了众多TileLink链路的TileLink交换器。在任何从端被接收的一条消息可能被转发到任何主端，反之亦然。因为，所有交换器的端口连接到同一个TileLink交换器代理。

而一个连接PCIe桥的TileLink，它有着一个主端和一个从端口，在任何从端口接收的TileLink消息不会触发到达主端的寄生消息。类似地，发往主端的消息，不会触发到达从端的寄生消息。因此，这样的桥实质上是两个包含在同一个RTL模块中的独立的代理。

在构建代理结构图，并使用有向无环的规则约束该图时，假设所有的代理都遵守我们前面几部分中所描述的转发规则，那么我们可以证明这样的TileLink拓扑网络是可以保证完全避免死锁的。能证明完全避免死锁的原因及证据不在本文中论述。

请求-响应消息排序

本节定义在一个特别强调具有多拍数据包的情况下，发送响应消息时的规则。在如下情况下，响应消息的第一拍可以出现在响应通道上：

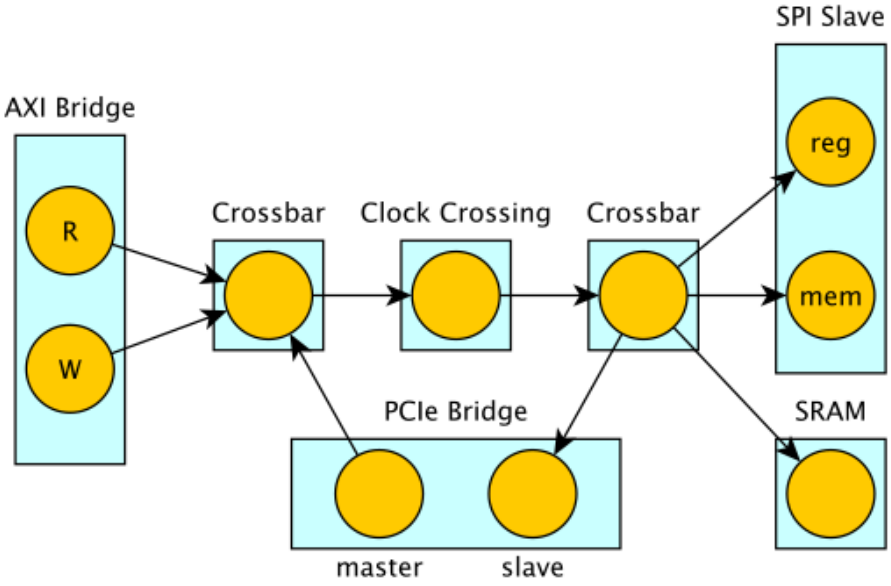

图 4.2: 一个TileLink网络的代理图。方块代表RTL模块，圆环代表代理

- 在请求消息被接收了的同一个周期出现，但不会在这周期之前出现。
- 在请求消息包中所有拍的数据都被接收完毕之前
- 在任意长的一段延迟之后

跟随在包的响应消息的第一拍后的多拍信号可以在任意延迟后出现，但与此同时，不能有来自其他消息的任一拍数据交错地插入此次传输中。

在一条请求消息的第一拍被接收时，一个回复消息可以同时被接收的实际情况(甚至可能在请求消息完成发送前)，与4.2.2部分的转发规则相互作用。这些规则规定了代理何时接受响应，例如在一个代理可以在d_valid为高时，将d_ready拉低。

例如一个设计者可能尝试实现一个主端接口，为了将一个同时发生的响应消息延迟到下一周期，接口在a_valid为高时保持d_ready为低。然而这样带来的通道D上的不确定的延迟是不被任何转发规则所允许的。甚至，一个TL-UL级别的从端接口可能将d_valid和d_ready分别连接到a_valid与a_ready。如此，非一致性的主端接口就引入了死锁的情况。

如果一个主端接口不能在请求消息的同一周期内接收响应消息，那么也可以在通道D输入的后面放置一个缓冲器。这个缓冲器可以保存一个同时发送的响应消息，并且直到它被填充之前，它都可以使得ready保持为高。这种响应处理逻辑使得在允许从端回复的速度尽可能快的同时，又满足了转发的规则。所有的代理都必须遵循如下规则，要么主动地去处理一个同步出现的回复消息，要么在接收端输入处放置一个缓冲器来保存该消息。 下文详细说明不同大小的包的请求及响应消息的互动情况:

簇发响应

图4.3描述了两个Get操作。Get操作的请求消息(操作码4)从通道A发出。它们都是访问32字节，在8字节的总线上分4拍拿走数据。我们能看到4拍的AccessAckData(操作码1)回复消息到达通道D。第一个回复消息在一个任意的延迟之后到达。主端接口必须要能够无限期地等待回复消息，因此在TileLink互联网络中不存在超时设定。最终，回复消息一定会到达，这是TileLink无死锁规则所能保证的。 第二个Get操作在请求消息被接收的同一个周期

内即被回复。这种重叠情况在Get第一拍的信号被接收之后是允许的。回复消息不能再提前出现：因为第二个Get首次出现时，a_ready为低，请求消息被拒绝了，所以d_valid信号也必须为低，否则就违反了第一个规则。

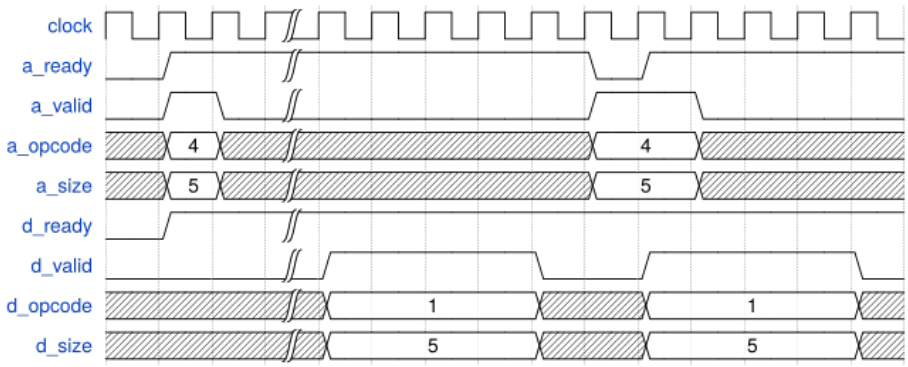

图 4.3: Get (4) 和 AccessAckData (1) 在一个8字节宽度总线上的最小与最大延迟

簇发请求

图4.4描述了两个Put操作。PutFullData请求消息(操作码0)在通道A发出，它们的AccessAck回复消息(操作码0)则在通道D返回。包的大小为32字节，四拍数据。与上小节不同的是，此次通道A发起请求消息。如名字所示，PutFullData消息带有数据载荷，而AccessAck消息则没有。

第一个AcessAck消息被延迟了任意的时间，但是请求者继续发送余下的请求消息。

第二个AccessAck消息在PutFullData消息的第一拍的同一周期内出现。这是我们的规则所能允许的最快的回复。如果a_ready或者a_valid信号任一个在该周期内为低，那么d_valid应该也为低。我们之前讨论的主端接口ready的告诫在此处适用：主端接口必须接收此时同时发送的AccessAck，甚至在它完成发送PutFullData消息之前。即使它可能会缓冲AccessAck消息，并且将其挂起，直到它完成发送请求消息。

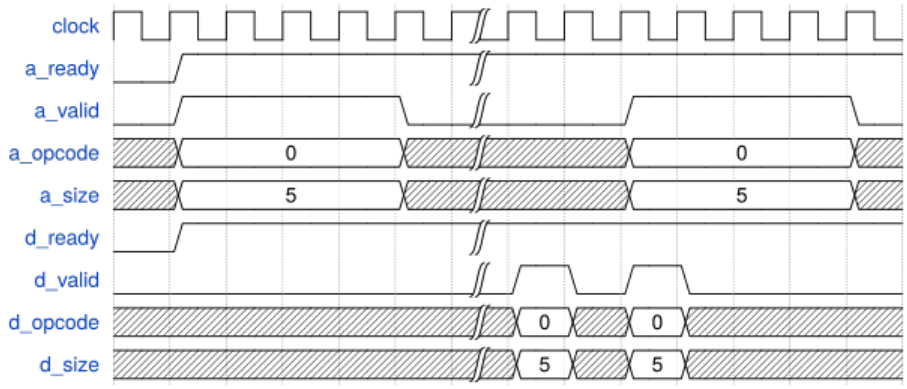

图 4.4: PutFullData (0) 和 AccessAck (0) 在一个8字节宽度总线上的最小与最大延迟

簇发请求与响应

都含有数据载荷的请求和响应消息适用的相同的规则，如下所示：

响应消息的第一拍不能出现在请求消息的第一拍之前，但是可以被任意的延迟。此外，响应消息各拍可被无限期延迟，且对于大部分这种类型的操作，延迟的时间至少与接收请求消息各拍信号所花费的时间相当。

图4.5描述了由一个分别都带有数据载荷的请求和响应消息所组成的原子性操作。对于大小为4，16字节(2拍)的操作，要么在请求和响应消息之间存在较长的延迟，要么两者的各拍信号重叠。如果它们需要相应的请求消息的各拍信号数据，那么响应各拍信号可能会被延迟。如果每条消息的整体都能放入一拍之中(2^3=8=1beat)，那么这些消息就能完全地重叠。

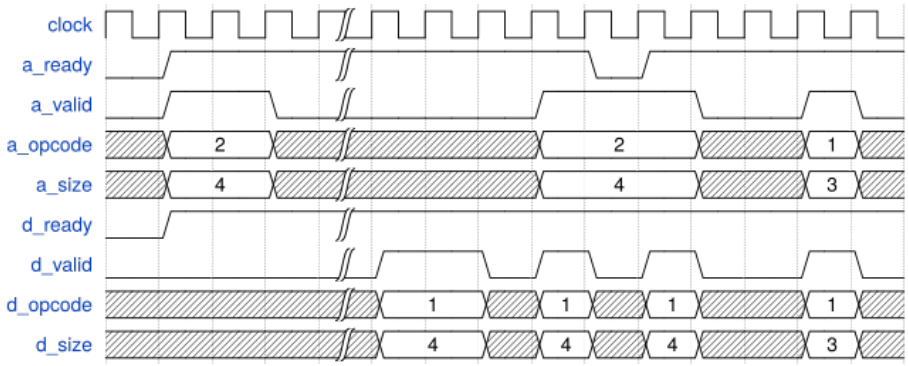

图 4.5: ArithmeticData (2) 和 AccessAckData (1) 在一个8字节宽度总线上的延迟

与旧式(legacy)总线的连接接口

旧式总线并不保证支持转发规则。当通过转接桥做协议转换，在等待传统总线接收一条消息时无限期地阻塞TileLink的通信，这将会违反TileLink的ready规则。因此，连接类似AXI这样的总线的桥必须有超时机制，以适应转发的向前推进规则的保护(4.2.2节)。如果现有总线在超时内，没有接收一个请求，那么这个请求必须被丢弃，并触发一个TileLink错误的响应。

如果传统总线要发送回复消息，转换桥必须要给其等待这些回复消息的时间长短设立一个限制，除非此总线被验证为是无死锁的。如果一个未验证的现有总线超过了时间限制，转换桥必须取消这个正在等待回复的请求，并插入一个TileLink错误响应，即使如果原始的现有总线发来的回复消息后来到达，也需要丢弃该消息。为了限制追踪丢弃回复消息的存储，可以使转换桥完全关闭一个死锁的传统总线。

在TileLink拓扑网络内部，会导致替代性消息产生的超时机制是被特别地禁止的。TileLink 代理等待其他TileLink 代理必须是无限期的。然而这并不妨碍TileLink 触发reset信号的看门狗机制。TileLink只有在所有代理都遵循本文档的规则时，才是无死锁的。如果对某个给定的拓扑网络的所有代理的实现的准确性没有信心时，我们可以使用看门狗。

错误

通道C和D包含了一个一位的错误信号域。这个域的使用取决于被操作码*_opcode标识的具体消息类型，在6-8部分中有具体描述。因为每条请求消息需要一个特定大小的回复消息。这个域使得一个代理在他探测到了数据不正确的情况下，创建一个回复消息。

错误域只能在包的最后一拍拉高，并且指示了相关的包内，任意一拍或者多拍数据出现了错误。不管最终是否出现了错误，整条消息的所有拍都必须全部发送。

字节通路

带有数据域的TileLink通道总是自然地以小端对齐方式运载数据载荷，如果数据总线宽度为_w_字节(必须为2的幂)，那么(address&！(w-1))就是0号字节通路的数据的地址。例如，总线宽度为16字节时，一个字节通路传输的数据的地址的最低一部分总是相同的，如4.7图中所示。

TielLink总是用对齐的地址来描述2的幂大小的字节范围。(address&((1<<size)-1)=0)总是保持。因此要么一个操作使用所有的字节通路，或者使用2的幂的部分。一个操作使用的字节通路称为有效字节通路。在4.7图中无效的字节通路被删去了。

在带有掩码(mask)域的通道A和B上，所有的无效字节通路的掩码位必须为低。而除了PutPartialData以外的消息，所有的有效字节通路的掩码的信号必须为高。

掩码信号也被其他不带数据载荷的消息使用，当操作需要使用的大小小于数据总线时，掩码应该要按带有数据载荷的信号时一样生成。对于比数据总线大小更大的操作，mask的所有位数都应该拉高，尽管消息仍为一拍，例如图4.6中所示。

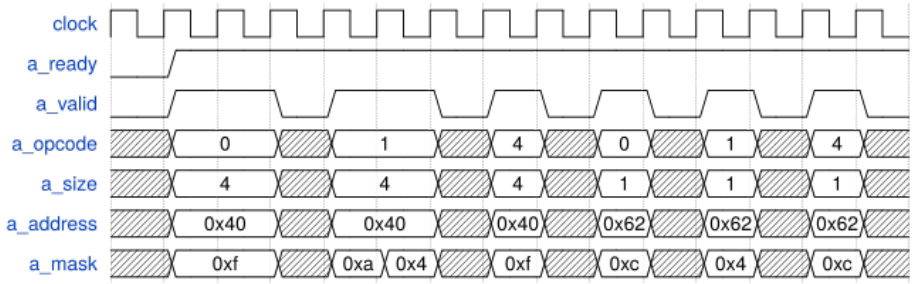

图 4.6: 字节通路掩码的实例。 PutFullData (0)必须将掩码设为全1，因此，第一个消息的所有拍都被有全1的掩码。相比而言，PutPartialData (1) 的各拍则可使用高或低的掩码。 Get (4) 消息不是簇发的，但仍然要将所有需要的字节通路设为高。 对于长度小于一拍的消息， 不需使用的字节通路应该将其掩码设为0（参见针对地址0x62的消息掩码设置）

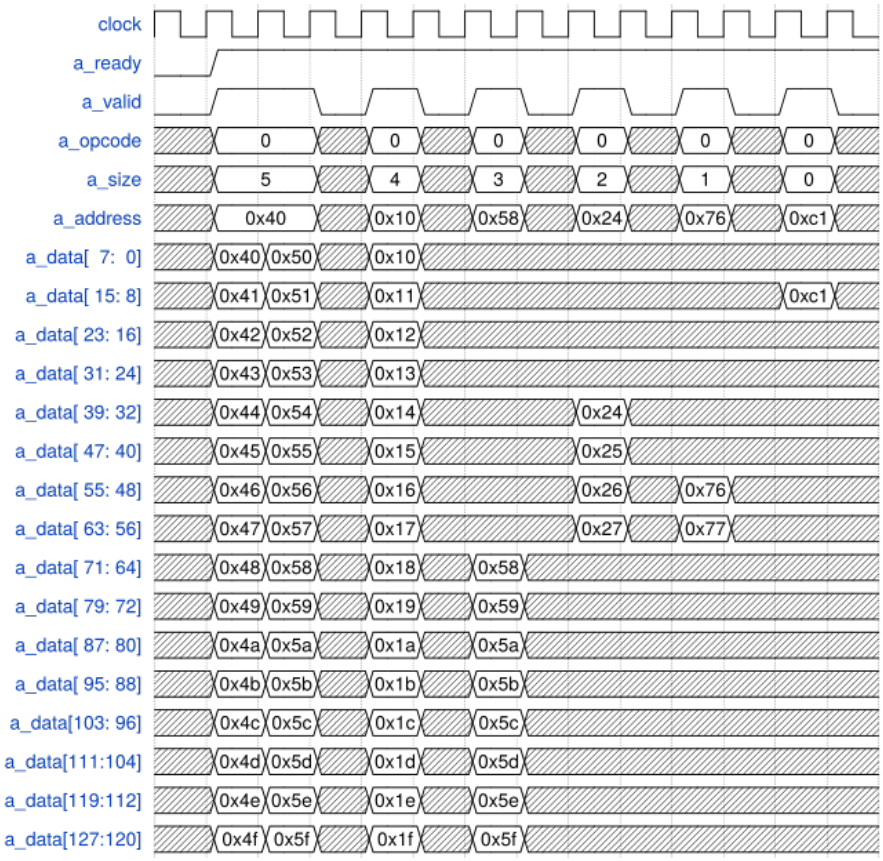

图 4.7: 在一个16字节宽度总线上的数据载荷分布。请注意，每个字节通路所对应数据载荷地址的低4位是固定的。同时，如果size小于数据总线的宽度，只有部分的字节通路会被使用。簇发操作会自动递增数据载荷的地址，但是控制信号将在整个簇发过程中保持稳定。

第 5 章

操作与消息

带有主端接口的 TileLink 代理通过执行各类操作与共享存储系统进行交互。操作会完成我们期待的对地址范围的修改，可能是数据值，可能是这段地址内数据的权限，也可能是在存储层次中的位置。操作通过在通道内传输的具体消息的交换来实现。为了支持一个操作的实现，我们需要支持组成该操作的所有消息。本章列出了所有 TileLink 的操作以及为了实现他们所需要交换的消息。我们详细叙述每个操作的具体的消息交换过程。其中操作分别三个兼容级别，TL-UL,TL-UH,TL-C。

操作分类

TileLink 操作能被分为下列三组：

- Accesses (A) 在具体的地址读或写数据。
- Hints (H) 只是提供一些信息，没有实际的影响。
- Transfers (T) 在拓扑网络中改变权限或移动缓存拷贝。

不是每个TileLink代理都需要支持所有操作。取决于其的TileLink 兼容级别，一个代理需要支持的操作如下表所列。

操作	类型	TL-UL	TL-UH	TL-C	目的
Get	A	y	y	y	从一个地址范围进行读取
Put	A	y	y	y	写入一个地址范围
Atomic	A	.	y	y	在一个地址范围进行读—修改—写操作（read-modified-write）
Intent	H	.	y	y	对未来可能进行的操作进行提前的预告
Acquire	T	.	.	y	缓存一份地址范围的内容的拷贝，或者提升一份拷贝的权限
Release	T	.	.	y	写回一个被缓存过的地址范围的拷贝内容，或者释放一份被缓存的拷贝的权限

表5.1: TileLink操作总览

消息分类

操作通过在五个 TileLink 通道内交换消息来执行。某些消息携带数据载荷，而有些则没有。如果一个 TileLink 消息携带数据载荷，那么其消息名以Data结尾。不是每一个通道都支持所有类型的消息。某些数据的到达必然会导致最终会有一个回复消息发送到请求发出者。带有回复的操作和消息分类如图5.1所示。表5.2通过兼容级别和操作对TileLink用到的消息进行分组。表5.3以通道和操作码来对消息进行排序。

注意到存在不同消息类型有着相同的操作码。不同的通道对于操作码有着不同的命名空间。在任何指定的通道内，每一个可能的消息类型有着独特的操作码。不管是在哪些通道内被进行交换的，相同类型的消息有着相同的操作码。操作码空间通过消息内容的有效解码进行分配。TileLink规范保留增加操作码的权利。

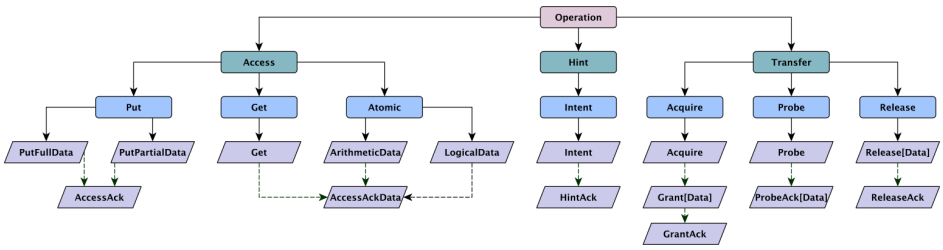

图 5.1: 所有的操作（蓝色）和他们的消息（紫色）。带点箭头表明了请求和响应的配对。TL-UL只要求支持Get和Put操作。TL-UH需要支持所有的Hint和Access操作。TL-C则需要支持所有的操作。

消息	操作	操作码	A	B	C	D	E	响应	兼容级别
Get	Get	4	y	y	.	.	.	AccessAckData	TL-UL
AccessAckData	Get or Atomic	1	.	.	y	y	.		TL-UL
PutFullData	Put	0	y	y	.	.	.	AccessAck	TL-UL
PutPartialData	Put	1	y	y	.	.	.	AccessAck	TL-UL
AccessAck	Put	0	.	.	y	y	.		TL-UL
ArithmeticData	Atomic	2	y	y	.	.	.	AccessAckData	TL-UH
LogicalData	Atomic	3	y	y	.	.	.	AccessAckData	TL-UH
Intent	Intent	5	y	y	.	.	.	HintAck	TL-UH
HintAck	Intent	2	.	.	y	y	.		TL-UH
Acquire	Acquire	6	y	.	.	.	.	Grant or GrantData	TL-C
Grant	Acquire	4	.	.	.	y	.	GrantAck	TL-C
GrantData	Acquire	5	.	.	.	y	.	GrantAck	TL-C
GrantAck	Acquire	-	.	.	.	.	y		TL-C
Probe	Probe	6	.	y	.	.	.	ProbeAck or ProbeAckData	TL-C
ProbeAck	Probe	4	.	.	y	.	.		TL-C
ProbeAckData	Probe	5	.	.	y	.	.		TL-C
Release	Release	6	.	.	y	.	.	rleaseAck	TL-C
ReleaseData	rlease	7	.	.	y	.	.	ReleaseAck	TL-C
ReleaseAck	release	6	.	.	.	y	.		TL-C

表5.2: 通过操作和兼容级别进行分组的TileLink消息总览

通道	操作码	消息	操作	响应
A	0	PutFullData	Put	AccessAck
	1	PutPartialData	Put	AccessAck
	2	ArithmeticData	Atomic	AccessAckData
	3	LogicalData	Atomic	AccessAckData
	4	Get	Get	AccessAckData
	5	Intent	Intent	HintAck
	6	Acquire	Acquire	Grant or GrantData
B	0	PutFullData	Put	AccessAck
	1	PutPartialData	Put	AccessAck
	2	ArithmeticData	Atomic	AccessAckData
	3	LogicalData	Atomic	AccessAckData
	4	Get	Get	AccessAckData
	5	Intent	Intent	HintAck
	6	Probe	Acquire	ProbeAck or ProbeAckData
C	0	AccessAck	Put	
	1	AccessAckData	Get or Atomic	
	2	HintAck	Intent	
	4	ProbeAck	Acquire	
	5	ProbeAckData	Acquire	
	6	Release	Release	ReleaseAck
	7	ReleaseData	Release	ReleaseAck
D	0	AccessAck	Put	
	1	AccessAckData	Get or Atomic	
	2	HintAck	Intent	
	4	Grant	Acquire	GrantAck
	5	GrantData	Acquire	GrantAck
	6	ReleaseAck	Release	
E	-	GrantAck	Acquire	

表5.3: 通过通道与操作码进行排序的TileLink消息

寻址

TileLink通道内运载的所有地址都是物理地址。从 TileLink 有向无环图中的节点出发，每一个有效地址必须导向一条唯一通往一个具体的从端的路径。在TileLink中，地址决定了哪些操作是可以执行的，哪些效果可以产生，哪些排序约束可以施加。能被添加为一个地址空间的属性包括：其兼容级别、存储一致性模型、可缓存性、FIFO 排序要求、可执行性、特权登记、以及任何服务质量保证。

例如，当主端在一个特定地址上执行一个操作时，它对这个请求是否能被缓存没有任何控制权，而这是由拓扑网络决定的。如果一个特定的从端，不能支持Get操作，那么位于主端及该从端之间的缓存不能缓存发往该从端地址的 Get 操作。类似的，如果一个从端不能支持Put操作，那么缓存至少要对发往该从端地址的 Put 操作进行写穿通。这些要求执行的具体机制不在本文叙述的内容之中。

我们希望一个片上系统的实现能创建一个本地地址映射，能用来描述具有副作用（由读写操作导致的非内存操作）的地址范围。这个映射能被缓存用于决定是否应该缓存一个特定的Get操作。类似地，一个交换器能使用这个地址映射来决定通过哪个端口来引导一个操作。

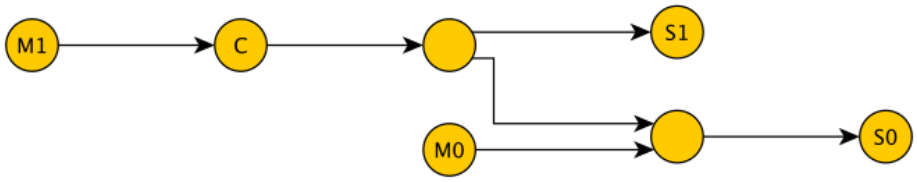

图 5.2: 一个简单的代理图。两个主代理M0和M1都可以访问从代理S0，但是只有M1可以通过缓存C访问从代理S1。

对于地址映射，我们希望它不只是一个单独的全局地址映射。因为，一旦在TileLink拓扑网络中发生的一个动作，就会使得地址映射的某些属性发生改变。例如，图5.2中主端 M1 能访问从端 S0 ， S1 ，同时 M1 只能访问从端 S0 。除了仅仅访问外，某些 TileLink 代理可能改变他们下级的从端的属性。例如，在图5.2中的缓存 C 可能导致从端 S1 的地址范围支持原子性操作，而原本这个从端是不支持的。

当无法提前知道一个给定地址范围从端设备具备哪些属性时，TileLink 拓扑网络其他部分必须清楚地定义目标从端是何种类型。例如，我们可以保守地假设，对外部的地址范围的一切操作都不支持 Get 和 Put 操作，或者可以乐观地认为，我们要求这个TileLink拓扑网络只能添加支持所有操作的设备。当我们面对一个不确定的 TileLink 从端端口时，这个端口必须要有文档来描述这个端口对应地址具备什么样的属性。类似地，当我们面对一个不清楚的 TileLink 主端端口时，该端口必须带有文档描述主端会对相关地址空间请求何种操作。

我们强烈建议如果一个地址区域不支持 Get 或者 Put ，那么这个地址区域应该被上对齐或者下对齐到最近的 4kB 的倍数。这会使得带有 TLB 的处理器更方便地处理地址映射。同样的理由适用于任何可能在未来定义的地址范围修改器。

显然，包操作不能越过不同属性从端的上下边界。因此从端不能支持比其要求的最小的对齐地址（通常我们希望至少为4kB）更大的包。主端则不能发出比从端所支持的尺寸更大的操作。然而，TileLink 中可以设计模块将较大数据块的操作分割成更小的能适应目标设备的操作。本地地址映射可能被使用，不过，主端要如何得到这些信息并不在本文档中定义。

为了优化吞吐量，可以追踪哪些地址范围是对请求是以 FIFO 的顺序进行回复。一般情况下，TileLink 中的回复是完全乱序的。但是，如果我们知道一个给定地址范围是按照FIFO的顺序进行回复，那么便能够直接将 TL-UL 级别的消息转换为 TL-UH 级别的消息。为以上这些原因，我们希望地址映射含有一个可选的FIFO域。所有共享一个 FIFO 域标识符的地址范围可以相互知道请求顺序，并按序回复。

此文档的未来的版本可能会定义：针对带有特定属性的地址范围的操作的行为。

源与目的地标识符

通道	目标	序列	路由信息	源标志	使用于
A	从	请求	a_address	a_source	d_source
B	主	请求	b_source	b_address	c_address

通道	目标	序列	路由信息	源标志	使用于
C	从	响应	c_address	.	
C	从	请求	c_address	c_source	d_source
D	主	响应	d_source	.	
D	主	请求	d_source	d_sink	e_sink
E	从	响应	e_sink	.	

表5.4: TileLink传输路径域总览

TileLink中并不是所有的消息都是根据地址来路由。尤其是，回复消息必须返回到正确的请求者。为支持此功能，TileLink通道包括了一个或更多的与本地链路相关的事务标识符域。这些域与address域一起用于路由消息，并且保证每一个发送中的消息能被特定操作唯一标识。表5.4提供了在每一个通道上用于引导请求和回复消息的的域的总结。

对于每一种请求消息来说，至少其有一个信号会被复制到与之对应的回复消息中。这些信号在表5.4中_源标志_列中标记出来了。例如，如果一个Get操作的a_source为4，那么AccessAckData回复消息的d_source也为4。成对的回复消息会基于对应的信号引导到相关的路径上。如表中_使用于_列。而其他某些信号也可能被要求在不同消息对间复制，下文及后续章节将有详细说明。

除了用于路由回复消息的路径外，事务标识符也能帮助每一条消息与正在进行中的操作相互对应地联系起来。标识符并不带有内在的语义，因此能被代理使用来标记一条消息和它的回复消息。当写入无存储转发代理、非阻塞的主端和从端时，这些标识符也十分有意义的。同样，标识符对于创建TileLink拓扑网络行为的监管器有用。

为了给代理追踪处于进程中的操作的状态的数量设立一个上限，我们在标识符上设立每个链路与通道唯一标识的约束。当一个已发出的请求仍在等待一个回复时，我们认为该请求使用的标识符正在处理。在链路中的每个通道中，每个正在处理的请求消息标识符必须是独一无二的。

通道A和C只基于地址路由它们的请求消息，但同时提供在通道D中回复所用到的source信号。因通道D的回复消息能基于d_opcode和d_source区分开来，我们允许它们从分离的命名空间来赋值source标识符。换而言之，进程中的A和C的请求消息能在a_source和c_source信号上使用相同的值，但是该值不能在当前请求仍处于进程中时，在各自通道内被再次使用。

因为通道C对通道B请求的回复仅路由到一个从端，并且可以用c_address唯一标识正在处理的操作，我们可以放松通道B上的请求区分度的约束，只要求b_source和b_address的结合在进程中是能够区分的。为了在相同的地址同时probe多个主端以及在多个地址probe相同的主端，这种放松是必要的。通道C的回复可能使用任何与发送者相关的c_source，不过这个信号是被忽略的。

通道D必须为正在处理的grant请求提供唯一的d_sink事务标识符。通道D的回复可能使用任何与发送者相关的d_sink，不过这个信号也是被忽略的。

可能的标识符的范围对于给定的 TileLink 链路来说是独立的。因此通道内source和sink的宽度在不同的link间可能有着很大的区别。例如，一个交换器可能连接到两个主端 M 和 N，主端M可能声明使用source 0-2，同时主端 N 使用source 0-1。交换器有着去往这两个主端不同的两条链路，所以链路的本地所有源标识符是无关的。为了让交换器能将消息从主端引导到从端，交换器必须将源标识符整合到发往从端的消息所使用的一个共用的命名空间。一种方法是将 M 的源限定为0-2，并且将 N 的源限定为4-5。然后，交换器就可以决定哪条回复消息去往哪个主端。一个代理在事务标识符上的映射完全由其实现方式定义。例如我们

例子中的交换器为了优化译码逻辑选择不使用 source 3。在一条链路内，所有通道source宽度是相同的。

操作排序

在一个 TileLink 拓扑网络中，在任意给定时间内会有多个操作在进程中。这些操作可能以任意的顺序完成。为了确保主端能在一个操作完成后，再执行其他操作，TileLink 要求从端在操作结束时及时发送一个回复消息。因此，如果处理器要确保两个写操作X 和 Y 的顺序被其他所有代理获取时都是一致的，那么处理器发送 X 的PutFullData后，必须等待等待AccessAck回复，在此之后，才发送 Y 的PutFullData。

TileLink 的从端，包括缓存，不能在Put操作确认前就将数据写回。唯一的约束是，一旦确认消息发出后，整个拓扑网络不能观察到过去的状态。这是因为在确认消息发出后，所有的被缓存的数据的拷贝都必须是已更新的。例如，对于一个Put操作，其他缓存要么被Probe现有的数据拷贝，要么通过PutFullData将消息前递给其他缓存，并且在确认原始的请求消息前，收集对应的回复消息。

发射回复消息的代理需要保证它们接收的操作是一个有效地序列。例如，假设一个代理接收了两个 Put，X和Y，并且都没有被确认。它必须选择一定的顺序，例如说是 X 在 Y 之前。如果选择了这样的顺序。必须保证只有三个状态， X 与 Y 前的状态， X 之后且 Y 之前的状态和 X 与 Y 之后的状态。代理不一定要以这样的顺序发射回复消息。然而，在代理已发射了一条回复之后，例如 Y ，如果此时接受了新的操作 Z ，那么 Z 必须排在 Y 之后。

这些规则确保每个代理看到的全局操作排序与主端的确认信号引导的局部排序是一致的。处理器可以等待发出的确认消息返回后再发起其他请求，来实现 fence 指令。这样的的能力使得多处理器在 TileLink 的共享存储系统中，安全地同步执行操作。

第 6 章

TileLink 无缓存轻量级(TL-UL)

TileLink 无缓存轻量级(TL-UL)是最简单的TileLink协议兼容级别。它可用于连接低性能的外设以减小总线的面积消耗。该兼容级别的代理都支持两种存储访问操作：

读(Get)操作：从底层内存中读取一定量的数据。

写(Put)操作：向底层内存中写入一定数目的数据。写操作支持基于字节t通路掩码的部分写功能。

这些操作都通过在4.3小节定义的两阶段请求/响应握手完成。在TL-UL中，每条消息都必须放在一拍中，不支持簇发操作。TL-UL一共定义了与存储访问操作相关的三种请求消息和两种响应消息类型。表6.1中列举了这些消息。

Message	编码	操作	A	D	响应
Get	4	Get	y		AccessAckData
AccessAckData	1	Get or Atomic		y	
PutFullData	0	Put	y		AccessAck
PutPartialData	1	Put	y		AccessAck
AccessAck	0	Put		y	

表 6.1: TL-UL 消息总览

数据流与波形

这部分的图提供了TL-UL的操作的波形图和消息序列表。图6.1描述了在一对代理之间读写(Get/Put)操作的波形。

下面用消息序列来展现在代理间发送消息的排序和依赖性，以及一段时间后它们所采取的回复行为。在消息序列中，时间轴冲下，顶部为早前的消息。图6.2展现了一对 代理间的读操作。图6.3展现了一对代理间的写操作。最后图6.4展现了在两级主从代理间采取前述操作的消息序列。

消息

这一部分定义了 TL-UL 所包含的五个消息类型的信号译码。

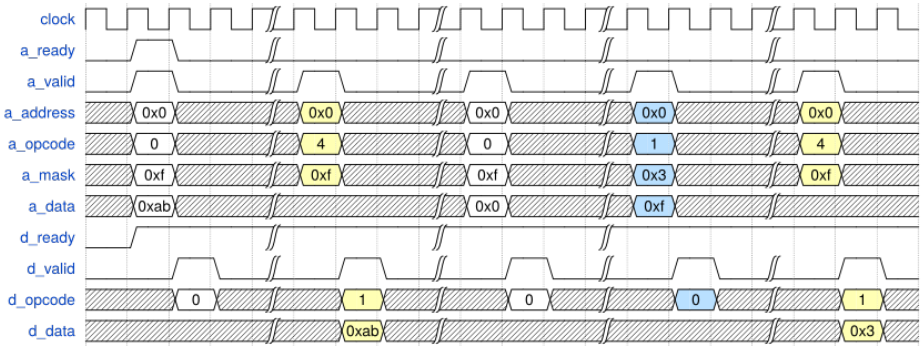

图 6.1: 包含Get和Put操作的波形。PutFullData写数据0xab， Get读回数据0xab，PutFullData写入0x0，PutPartialData写入0x3，最后Get读回0x3

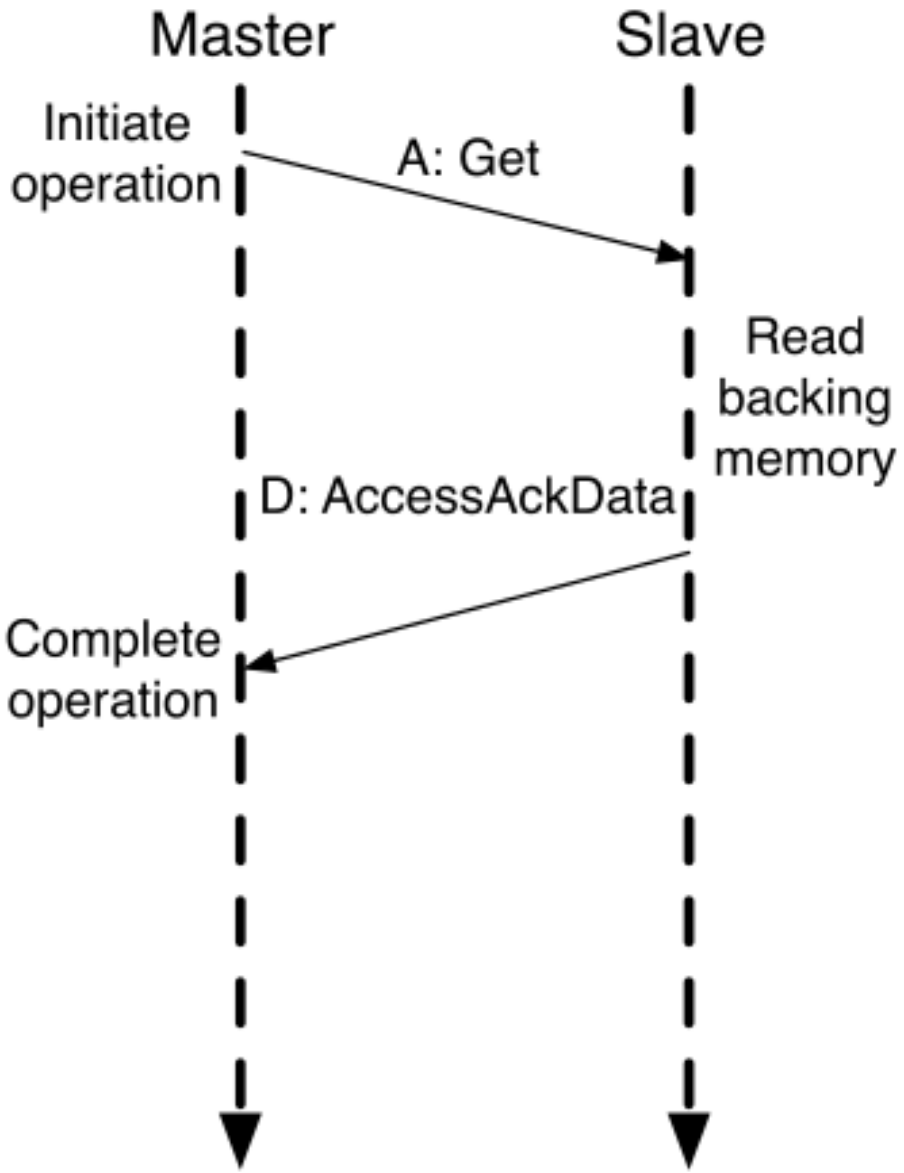

图 6.2: 读操作的消息序列。主端向从端发出一个Get报文。收到报文并读取数据后，从端响应主端发回一个AccessAckData报文

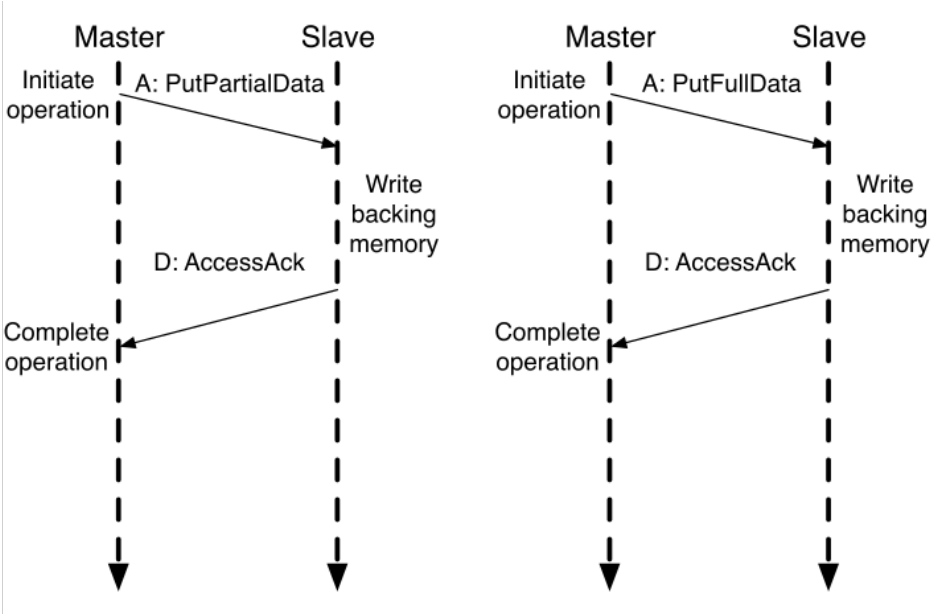

图 6.3: 写操作的消息序列。主端向从端发出一个PutPartialData或PutFullData报文。在写入数据后，从端响应主端发回一个AccessAck报文

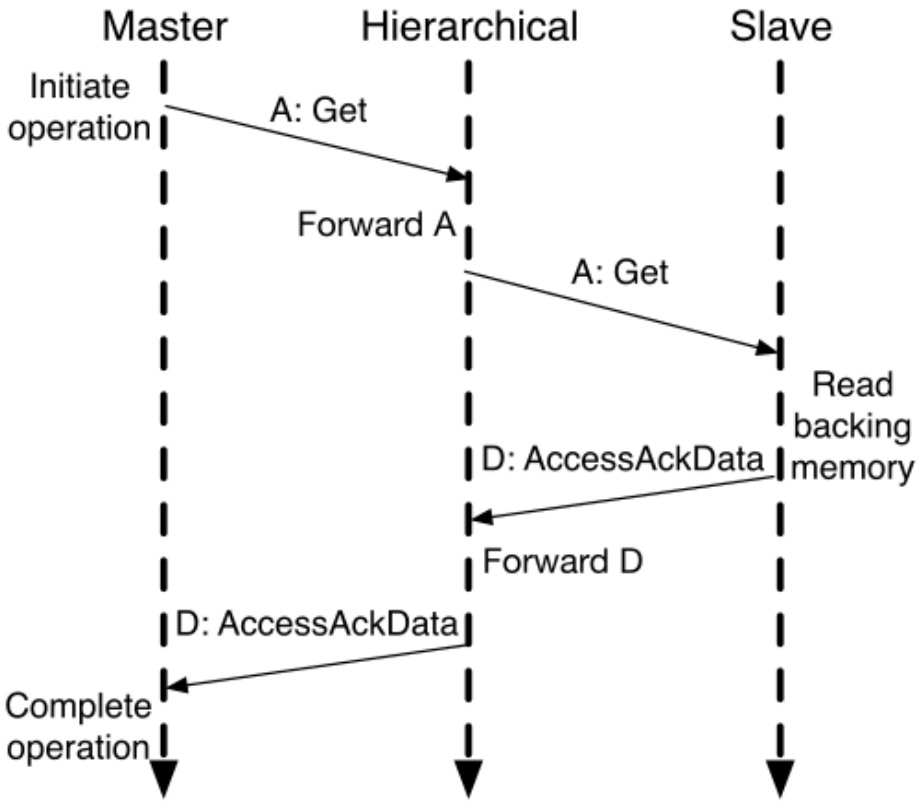

图 6.4: 穿越多个代理层级的数据块读取消息序列。层级分界上的代理负责将Get报文转发至外部的从代理和将AccessAckData响应报文转发给主代理

读(Get):

Get 消息是代理发出的请求报文，用于在读取数据时，访问一块特定的数据存储块。表6.2展现了通道 A 内，该消息的信号译码。

a_opcode 必须是Get，编码为4。

a_param 为保留位，为之后的hints操作预留，必须为0。

a_size 表明了请求代理想要读取的数据量，按照log2(byte)计算。a_size也代表了回复消息中AccessAckData的尺寸，并非Get操作特有。在 TL-UL 中a_size 不能大于实际数据总线的物理宽度。

a_source 是发出请求的主端的标识符。它会被从端代理拷贝（至回复报文中），以确保回复报文的正确路由（见5.4节）。

a_address 必须与a_size对齐。

a_mask 选择需要读取的部分字节通路（见4.6节）。a_size、a_address和a_mask的取值需要依赖。一个正确的Get请求的mask必须选择一个合适宽度的连续子数据通路。

a_data 在 TL-UL 中被忽略，可能取任何值。

通道 A	类型	宽度	编码
a_opcode	C	3	必须是Get(4)
a_param	C	3	保留位，必须为0
a_size	C	z	将要被从端读取和返回的2^n字节
a_source	C	o	发射该请求的主端源标识符
a_address	C	a	访存的目标地址，以字节形式
a_mask	D	w	将要读取的字节通路
a_data	D	8w	被忽略的，可能取任何值

表 6.2: Get消息的编码

完整写(PutFullData):

PutFullData 代理请求访问并写入一整块数据时发出的消息报文。第7章将说明为完整写(全使能的mask)加入一个单独的操作码的意图。表6.2说明了此消息在通道A中的编码。

a_opcode 必须为 PutFullData，编码为0。

a_param 为保留位，为之后的hints操作预留，必须为0。

a_size 表明了请求代理想要写入的数据量，按照log2(byte)计算。a_size也代表了请求消息的尺寸。在 TL-UL 中a_size 不能大于实际数据总线的物理宽度。

a_source 是发出请求的主端的标识符。它会被从端代理拷贝（至回复报文中），以确保回复报文的正确路由（见5.4节）。

a_address 必须与a_size对齐，a_address到a_address+2**（a_size-1）的整个内容都会被写。

a_mask 选择需要读取的部分字节通路（见4.6节）。a_mask中的一个高比特代表一个字节被写。 a_size、a_address和a_mask的取值需要依赖。PutFullDate必须使用一个连续的掩码(mask)。如果a_size等于或大于数据总线的物理宽度，掩码必须为全高。

a_data 是实际写入的数据载荷。a_data的任一没有被掩码选择的字节都会被忽略，可能取任意值。

通道 A	类型	宽度	编码
a_opcode	C	3	必须为PutFullData(0)
a_param	C	3	保留位，必须为0
a_size	C	z	2^n 字节将会被从端写入
a_source	C	o	发出该请求的主端的源标识符
a_address	C	a	访存的目标地址（字节）
a_mask	D	w	被写入的字节通路，必须是连续的
a_data	D	8w	被写入的数据载荷

Table 6.3: PutFullData消息的编码

部分写(PutPartialData)：

PutPartialData 代理请求访问并写入一块数据时发出的消息报文。PutPartialData被用于写入在任意字节上的数据。表6.4展示了此消息在通道A中的编码。

a_opcode 必须为 PutPartialData，并且编码为1。

a_param 为保留位，为之后的hints操作预留，必须为0。

a_size 表明了请求代理想要写入的数据量，按照log2(byte)计算。a_size也代表了请求消息的尺寸。在 TL-UL 中a_size 不能大于实际数据总线的物理宽度。

a_source 是发出请求的主端的标识符。它会被从端代理拷贝（至回复报文中），以确保回复报文的正确路由（见5.4节）。

a_address 必须与a_size对齐，a_address到a_address+2** (a_size-1) 的内容的一部分会被写。

a_mask 选择需要读取的部分字节通路（见4.6节）。a_mask中的一个高比特代表一个字节被写。 a_size、a_address和a_mask的取值需要依赖。PutPartialData 可能写入比a_size所标识更少的数据，取决于 a_mask 的内容。a_mask 的任何的高比特都必须在 a_size 的对齐区域内，但是高比特无须连续。

a_data 是被实际写入的数据载荷，a_data 的任意没有被 a_mask 覆盖的子集都会被忽略，可能取任意值。

通道 A	类型	宽度	编码
a_opcode	C	3	必须为PutPartialData(1)
a_param	C	3	保留位，必须为0
a_size	C	z	2^n字节会被从端写入
a_source	C	o	发射该消息的主端源标识符
a_address	C	a	访存的目标地址（字节）
a_mask	D	w	被写入的字节通路
a_data	D	8w	被写入的数据载荷

Table 6.4: PutPartialData消息的编码

无数据确认(AccessAck)：

AccessAck是一个送往原请求代理的无数据确认消息。表6.5展示了此消息在通道C中的编码。

c_opcode 必须为 AccessAck，编码为0。

c_param 保留为TL-C的操作码，（在TL-UL中）必须指定为0。

c_size 包括了访问的数据的尺寸，尽管这条消息本身不包含任何数据。尺寸和地址需要对齐。c_address必须和请求中的b_address配对。

c_source 发出该回复的代理的ID。

c_address 操作的目标地址。

c_data 被忽略，可能被指定为任意值。

c_error 标识在主端尝访问存储过程中发生的错误。

通道 C	类型	宽度	编码
c_opcode	C	3	必须为AccessAck(0)
c_param	C	2	保留，但必须为0
c_size	C	z	2^n字节会被从端访问
c_source	C	o	发出该回复的主端的源标识符
c_address	C	a	目标地址
c_data	D	8w	忽略，可以是任意值
c_error	F	1	标识主端不能处理的请求

表 6.5: AccessAck消息的编码

带数据确认(AccessAckData)：

AccessAckData是一个向原请求代理返回数据的确认消息。表6.6展示了此消息在通道C中的编码。

c_opcode 必须为 AccessAckData，编码为1。

c_param 保留为TL-C的操作码，（在TL-UL中）必须指定为0。

c_size 包括了访问的数据的尺寸，对应该回复所携带数据的尺寸。尺寸和地址需要对齐。

c_source 发出该回复的代理的ID。

c_address 操作的目标地址。

c_data 被访问的数据。在簇发中，该域可变化。
c_error 标识在主端尝访问存储过程中发生的错误。

通道 C	类型	宽度	编码
c_opcode	C	3	必须为AccessAckData(1)
c_param	C	2	保留，但必须为0
c_size	C	z	2^n字节会被从端访问
c_source	C	o	发出该回复的主端的源标识符
c_address	C	a	目标地址
c_data	D	8w	数据载荷
c_error	F	1	标识主端不能处理的请求

Table 6.6: AccessAckData消息的编码

第 7 章

TileLink 无缓冲重量级 (TL-UH)

TileLink 无缓冲重量级(TL-UH)是用于末级缓存之外的总线的。这种应用中不需要使用权限转换的操作。它建立在TL-UL的基础上，并提供一部分额外的操作。

原子(Atomic)操作 在原子性地读取现存的数据值的同时，同步地写入一个新的值，此新值为某些逻辑和算法操作的结果。

预处理(Hint)操作 提供了与某些性能优化相关的可选的暗示性消息。

簇发(Burst)消息 允许带有比数据总线宽度更大的数据的消息在多个周期内作为数据包传输。应用于在Get、Put和原子操作中多种包含数据的消息。

原子性操作允许代理访问特定的数据块来实现如此的一个存储操作：在原子性地读取现存的数据值的同时，同步地写入一个新的值，此新值为某些逻辑和算术操作的结果。每一个操作使用两个操作值；其中一个是被原子性消息携带的数据，第二个是目标地址已有的现有值。这个操作返回一个逻辑数据给请求者。鉴别逻辑操作和算术操作是有意义的，因为实现这两种操作对ALU的要求是有着显著的不同的。

预处理操作作为实现可选的性能优化的实现的一个技巧。他们不会修改数据的值，然而会引起代理去改变在具体数据块上可用的权限。由一个预处理提供的信息总是可以被任何接收到的从端安全地忽视掉，尽管该接收者必须发送一个确认消息。

包消息允许各类操作可以瞄准更大的地址范围，并且特别地允许带有数据载荷的消息的数据的尺寸可以大于数据总线的物理宽度。在Get、Put和原子中的任何不同类型的包含数据的消息都可能成为一个包。数据包没有新的消息类型，而第六章中的具体信号约束被移除了。在4.1和4.3节可以看到关于带有包的操作如何进行序列化的细节。

新的操作通过使用4.3节中出现的成对的请求和响应消息对来完成。总的来说，TL-UH相对于TL-UL增加了三个请求消息和一个响应消息。表7.1列举了这些消息。

消息	操作码	操作	A	D	响应	兼容级别
Get	4	读	y	.	AccessAckData	TL-UL原有
AccessAckData	1	读或原子	.	y		TL-UL原有
PutFullData	0	写	y	.	AccessAck	TL-UL原有
PutPartialData	1	写	y	.	AccessAck	TL-UL原有
AccessAck	0	写	.	y		TL-UL原有
ArithmeticData	2	原子	y	.	AccessAckData	TL-UH新添
LogicalData	3	原子	y	.	AccessAckData	TL-UH新添
Intent	5	预处理	y	.	HintAck	TL-UH新添
HintAck	2	预处理	.	y		TL-UH新添

Table 7.1: TL-UH 消息总览

消息流图和信号波形图

这一节的图提供了每一个额外增加的TL-UH操作的波形以及消息队列表。图7.1展现了在一对代理之间的原子和预处理操作的波形图。图7.2则展示了在一对代理之间，原子操作的消息流。图7.3展现了一群代理之间预处理操作的消息流。

簇发消息的波形图可以参考4.1和4.3节。

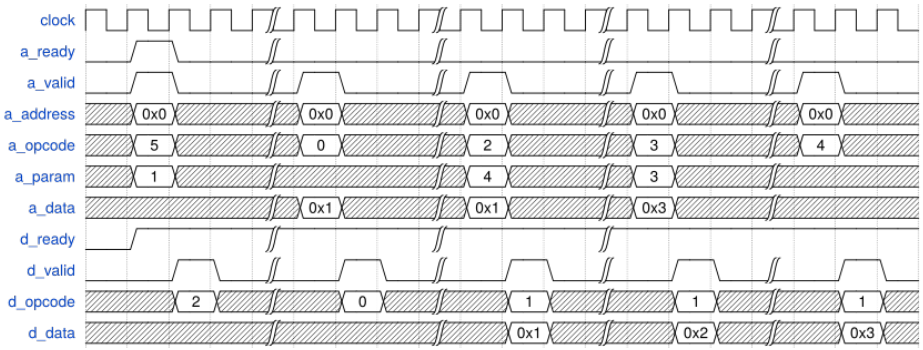

图 7.1: 原子和预处理操作的信号波形图。为写操作准备的预取，写入0x1，原子操作加0x1并返回0x1，原子操作交换0x3和0x2，读取最后的0x3

消息

这一节定义了组成TL-UH内的四种消息类型的所使用的信号的编码： ArithmeticData、LogicalData、Intent和HintAck。

算术数据(ArithmeticData)

一个算术数据消息是一个代理为对一数据块进行算术操作，先读取然后改写，而发起的访问一块特定的数据块的请求消息。表7.2展现了通道A用于这条消息的信号的编码。

a_opcode 必须为 ArithmeticData，编码为2。

a_param 用来具体制定要进行的原子性算术操作。被支持的算术操作组列在表7.3中，它包括了{MIN,MAX,MINU,MAXU,ADD}，代表了有符号和无符号整数的最大功能和最小功能，以及整数加法。

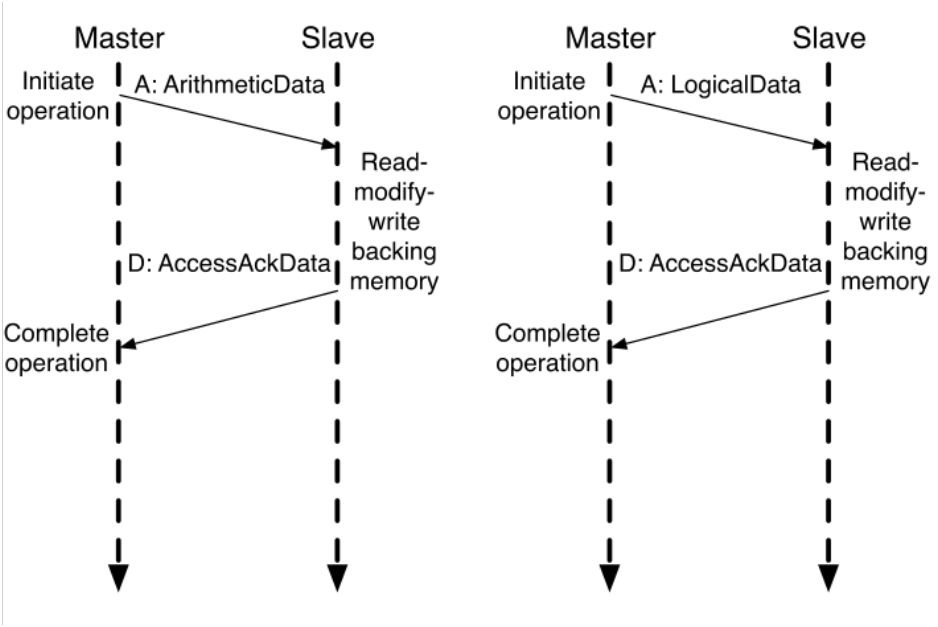

图 7.2: 原子内存访问的消息流

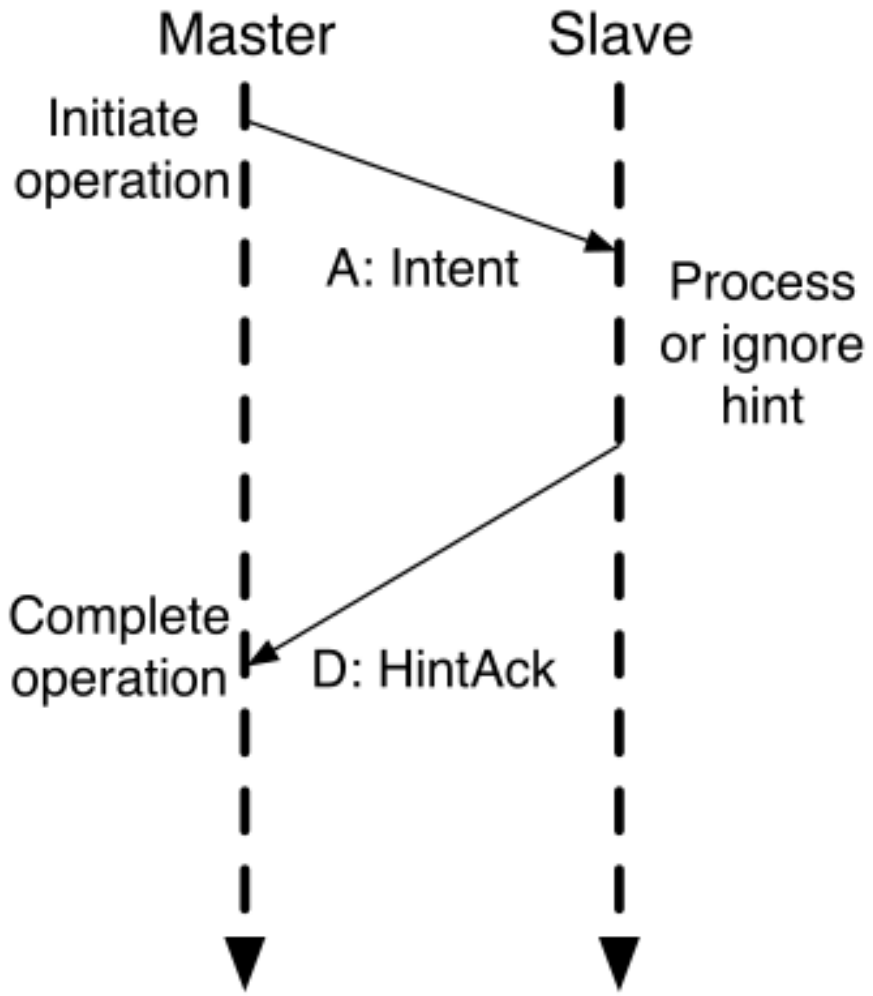

图 7.3: 预处理操作的消息流

a_size 是操作数的尺寸，按照log2(byte)计算。它反映了请求和响应报文中数据的尺寸。

a_source 是发出请求的主端的标识符。它会被从端代理拷贝（至回复报文中），以确保回复报文的正确路由（见5.4节）。

a_address 必须与 a_size 对齐。

a_mask 选择需要读取并改写的字节通路（见4.6节）。a_mask的每一比特与数据的一个字节相对应。a_size、a_address和a_mask的取值需要依赖。mask必须选择一个对齐的连续字节通路子集。

a_data 包含了算术操作的一个操作数（另一个可以在目标地址找到）。a_data的任何没有被a_mask选中的字节都会被忽略，可以取任何值。

通道 A	类型	宽度	编码
a_opcode	C	3	必须为 ArithmeticData(2)
a_param	C	3	见表7.3
a_size	C	z	2^n个字节会被从端读取或写
a_source	C	o	发射请求的主端源标识符
a_address	C	a	访问的目标地址，以字节形式
a_mask	D	w	要被读与写的字节通路
a_data	D	8w	要被用作操作数的数据载荷

表 7.2: ArithmeticData消息的编码

计算	编码	具体运算
MIN	0	写下两操作数的有符号最小值，并返回旧值
MAX	1	写两操作数的有符号最大值，并返回旧值
MINU	2	写两操作数无符号最小值，并返回旧值
MAXU	3	写两操作数无符号最大值，并返回旧值
ADD	4	写两操作数的和，并返回旧值

表 7.3: ArithmeticData的param域

逻辑数据(LogicalData)

一个算术数据消息是一个代理为对一数据块进行位逻辑操作，先读取然后改写，而发起的访问一块特定的数据块的请求消息。表7.4展示，这个消息内通道A内所使用的信号的编码。

a_opcode 必须为 LogicData，编码为3。

a_param 具体说明了要进行的原子性按位逻辑操作。支持的逻辑操作组列在表7.5中。它包括{XOP,OR,AND,SWAP}，代表了字节逻辑xor，or，and以及一个简单的操作数的交换。

a_size 是操作数的尺寸，以log2（byte）的形式。它反映了request的数据和 AccessAckData 回复的的尺寸。

a_source 是发出请求的主端的标识符。它会被从端代理拷贝（至回复报文中），以确保回复报文的正确路由（见5.4节）。

a_address 必须与 a_size 对齐。

a_mask 选择需要读取并改写的字节通路（见4.6节）。a_mask的每一比特与数据的一个

字节相对应。a_size、a_address和a_mask的取值需要依赖。mask必须选择一个对齐的连续字节通路子集。

a_data 包含了算术操作的一个操作数（另一个可以在目标地址找到）。a_data的任何没有被a_mask选中的字节都会被忽略，可以取任何值。

通道 A	类型	宽度	编码
a_opcode	C	3	必须为 LogicalData(3)
a_param	C	3	见表 7.5
a_size	C	z	2^n 个字节会被从端读写
a_source	C	o	发射请求的主端源标识符
a_address	C	a	访存目标地址，字节形式
a_mask	D	w	要被读写的字节通路
a_data	D	8w	要被写的数据载荷

表 7.4: LogicalData 消息的编码

计算	编码	具体运算
XOR	0	对两个操作数进行按位异或操作并写结果，返回旧值
OR	1	对两个操作数进行或操作并写结果，返回旧值
AND	2	对两个操作数进行与操作并写结果，返回旧值
SWAP	3	交换两操作数并返回旧值

表 7.5: LogicalData的param域

预处理(Intent)

一个Intent消息是一个代理为了表示未来的可能要访问一块特定数据块的目的而发出的请求消息。表7.6展现了通道A该消息所使用的信号的编码。

a_param 指定了此Hint操作所传递的具体意图。应注意，它的效果作用于从接口和从接口所连接的所有更外层节点。表7.7列出了所支持的intention的集合。它由{ PrefetchRead, PrefetchWrite }组成，表示为了读取的预取和为了写的预取。

a_size 该预取所需要的数据的大小

a_address 必须与a_size对齐.

a_mask 标识了intention适用的字节（4.6节）。a_size，a_address，和a_mask都需要互相对应。

a_data 被忽略并且可以取任何值

通道A	类型	宽度	编码
a_opcode	C	3	必须为Intent (5)
a_param	C	3	暗示性编码，可见表7.7
a_size	C	z	2^n个字节适用于该预取
a_source	C	o	发射该请求的主端源标识符
a_address	C	a	预取的目标地址，以字节形式
a_mask	D	w	预取使用的字节通路
a_data	D	8w	被忽略，可以是任何值

表 7.6: Intent消息域

类型	编码	具体作用
PrefetchRead	0	发送请求的代理想要读取目标数据
PrefetchWrite	1	发送请求的代理想要写目标数据

表 7.7: Intent消息的param域

预处理确认(HintAck)

HintAck 是用于一个Hint操作的确认消息。表7.8展示了通道D用于该消息的信号的编码。

d_opcode 必须为HintAck，编码为2。

d_param 保留并且必须指定为0。

d_size 包含了预取有关的数据的尺寸，尽管这个消息本身并不包含任何数据。

d_source 是从请求消息中的a_source得到的并且可以用于引导回复消息到正确的目的地（5.4小节）。

d_sink 被忽略，并且可以取任何值。

d_data 被忽略，并且可以取任何值。

d_error 表示了当从端试图进行一个操作时发生了一个错误。

通道D	类型	宽度	编码
d_opcode	C	3	必须为HintAck(2)
d_param	C	2	保留，必须为0
d_size	C	z	与Hint相关为2^n字节
d_source	C	o	接收该回复消息的主端源标识符
d_sink	C	i	忽略，可以是任何值
d_data	D	8w	忽略，可以是任何值
d_error	F	1	从端无法服务该请求

表 7.8: HintAck消息域

簇发(burst)消息

簇发消息可以包含比数据总线的物理宽度更大的数据。在一个周期内发送的数据的子集称为一个拍(beat)。簇发消息可能是任何在Get、Put以及原子等包含数据的操作内的不同消息。

4.5节给出了一个簇发消息序列化的细节。在表3.2中所定义的通道信号类型中，有三种是通过是否能在一个簇发消息的不同拍之间跳变来区分的。数据类型的信号（例如data, mask）允许在簇发的每一拍改变。终结类型的信号（例如error）只允许在簇发的最后一拍改变。控制类型的信号（例如address,size,param）必须在整个簇发过程中保持不变。4.3小节讲述簇发请求和响应报文的报文顺序问题。

定义PutFullData操作码的目的是允许代理针对全写掩码做性能优化。如果一个PutFullData消息是一个簇发消息，一个被优化的代理就可以通过第一拍判别整个簇发的掩码，而无需等待整个簇发的完成。

第 8 章

TileLink 缓存支持级 (TL-C)

TileLink中，TileLink缓存支持级(TL-C) 通过给主端代理提供缓存共享数据块副本的能力。依据实现过程中所定义的一致性协议，这些本地副本可以保证一致性。本章描述TL-C中缓存的数据副本上允许进行哪些访存操作，以及用来传输数据块的缓存的消息类型。实现中所定义的一致性协议描述了副本和权限如何通过具体的 TileLink 代理网络传输以回复所接收的存储访问操作。具体一致性协议的描述不在本文内容范围内。 总的来说TL-C在TileLink协议规范中新添加了如下内容：三种操作，三个通道，一个五步的消息序列模板以及十种消息类型。

新的操作 transfer(转换操作) 可以创建或者清除数据块的缓存副本。转换操作不会修改数据块的值，但是会改变他们的读写权限。 转换操作可以与之前定义的TL-UL TL-UH访存操作无缝协作，保证操作进行序列化的正确性。因为每一个转换操作逻辑上发生在另一个操作之前或之后，所有的代理都遵循该顺序，TileLink网络可以保证这种不变性(coherence invariant)。

在TileLink网络传输一个访存操作时，一个间质性的缓存(interstitial cache)能在其内部嵌套一个寄生转换操作。 缓存首先采用一个转换操作以取得缓存块的足够权限，并将其满足一致性的本地副本响应给存储访问操作发起者。

可缓存性是存储地址范围的一个属性，一个TileLink的实现需保证避免无法缓存地址范围的副本出现(第5.3章)。相反，在TL-UL和TL-UH中定义的访存操作可能被主端用来访问具有可缓存性的地址，而不需要自己将其缓存。某些主端需要缓存特定的数据块，而位于同内存储层级的其他主端可能选择不缓存。

下一节将概述在特定的、与实现相关的一致性协议中，设计人员可用的基本操作、消息和权限。本规范并不强制使用哪种一致性协议，而是为支持这些协议构建了公共的基础。

使用 TileLink 实现缓存一致性

所有基于Tilelink的一致性协议都由一系列操作组成，通过这些操作，可以完成读写数据块权限的在转变。在代理对已缓存的副本执行响应的操作之前，必须获得正确的权限。当代理希望在本地处理一个访问操作时，它必须首先使用转换操作来获得必要的权限。转换操作在网络上创建或删除副本，从而修改每个副本可以提供的权限。

代理中数据块副本的基本权限可以包括：None、Read或Read+Write。缓存架构中副本可用的权限取决于缓存层次结构中副本的当前存在情况，如下所述。

对于任何给定的地址，在给定主端和拥有该地址范围的从端之间都只会存在一个具体的路径。在TileLink网络DAG中，所有这些路径都覆盖会形成一个树，根节点上仅有一个从端

节点。对于每个地址，此树包含所有针对该地址的操作执行的路径。如果我们省略了所有不能缓存数据的代理，那么就会留下一个描述所有可能缓存该地址数据的缓存代理位置的树。

在逻辑时间的任意时刻，这些代理的某些子集真正包含缓存数据的副本。这些代理形成了 一致性树 。包容性(Inclusive)的TileLink一致性协议要求树在响应内存访问操作时进行提权(Grow)和降权(Shrink)操作。图中的每个节点都属于树中的位置，分为以下四类：

Nothing: 当前没有缓存数据副本的节点，没有读写权限。

Trunk: 主干：在尖端(Tip)和根(Root)之间的路径上拥有缓存副本的节点。具有其副本的读权限，该副本可能包含脏数据。

Tip: 顶点：具有缓存副本的节点，且可以用作内存访问序列化。对其拥有的可能包含脏数据的副本具有读写权限。

Branch: 分支：该节点由Trunk结点分出，具有只读数据缓存副本。

图8.1例举了几个覆盖在单个TileLink网络上的一致性树。在A中，树的根节点有唯一的副本，这使得它既是树的根节点，也是树的顶端节点。在B中，主端通过提升主干的特权(Grow)，获得写+读权限，直到它位于顶点处。在C中，另一个主端通过扩展一个分支获得了读权限，这意味着之前的顶点现在也是一个只读分支，且公共的父节点是主干节点。在D中，其他主端也升级为分支，进一步将尖端向根部移动，而最初的请求者已经主动修剪了它的分支。

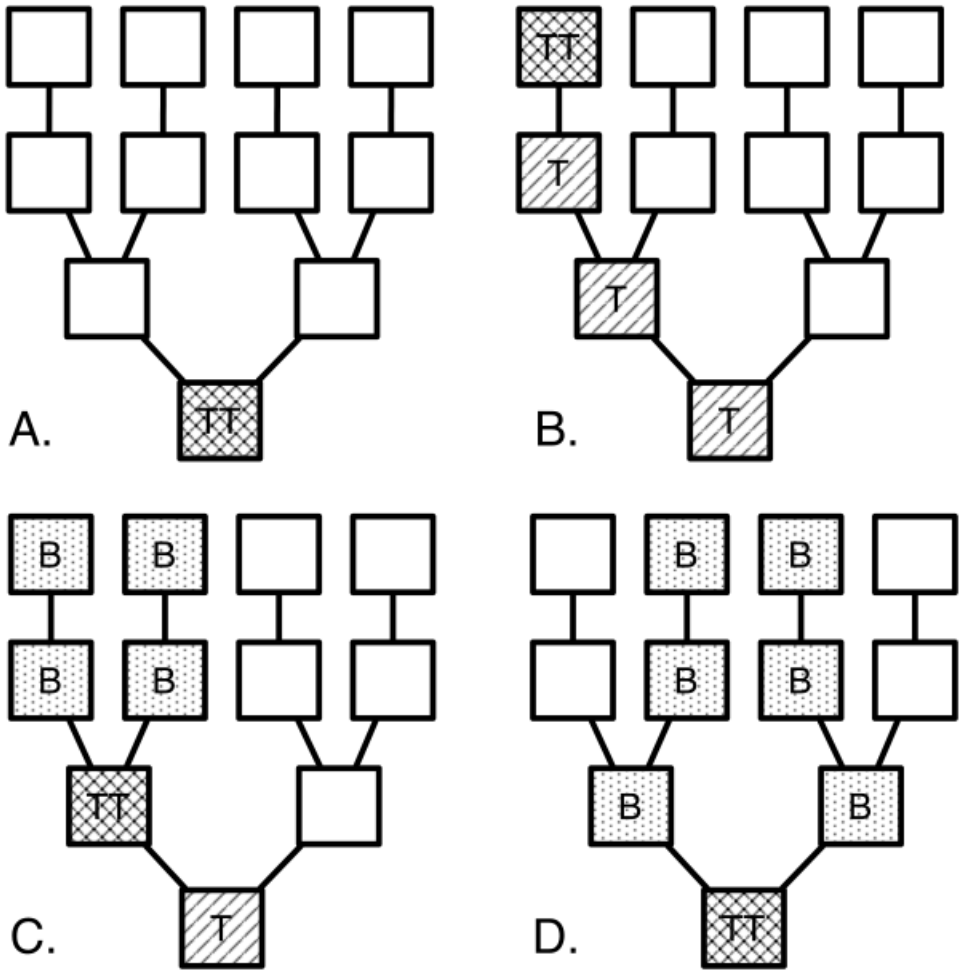

图 8.1: 同一个TileLink拓扑图中几种可能的一致性树。“TT”代表Trunk Tip，“T”代表Trunk，“B”代表Branch。A. 只有根节点拥有副本的读写权限。B. 在TT上只有一个主端拥有读写权限。C. 在B上有多个主端拥有读权限。D. 在B上又多个主端拥有读权限，有些B已被修剪

表8.1分别描述哪种状态下的节点上可以执行哪些访问操作，并且这还定义这些状态在一致性树中的位置，另外，协议具体定义的状态可以基于这些基本状态。

权限	支持的访问
None	None
Branch	Get
Trunk	Get
Tip	Get, PutPartial, PutFull, Logical, Arithmetic

表8.1: 权限与访存操作之间的关系

Operations

这三个新操作统称为transfer operations(第5章)，因为它们将数据块的副本传输到内存层次结构中的新位置

Acquire: 在请求主端中创建块(或其特定权限)的新副本。

Release: 从请求的主端将块的副本(或它的特定权限)释放回从端。

Probe: 强制将块的副本(或它的特定权限)从主端移到发起请求的从端。

Acquire操作要么以扩展树干的形式，要么以从现存的分支或者尖端添加新的分支的形式来拓展树。在新的分支生成前，旧的主干或分支可能需要递归的(recursive)Probe方法进行修剪。为了响应缓存容量冲突，可通过Release 操作主动裁剪分支。

Channels

为了支持转换操作，TL-C在执行内存访问操作所需的两个基本通道上添加了三个新通道。A和D通道也被重新用于发送额外的消息，以实现转换操作。转换操作使用的五个通道分别是：

通道A. 主端为了读取或写入缓存块副本而发起对权限的请求。

通道B. 从端查询或修改主端对缓存数据块的权限，或将内存访问前递(forward)给主端。

通道C. 主端响应通道B传输的消息，可能会释放带有任脏数据块的权限。也用于主动回写脏的缓存数据。

通道D. 从端向原始请求者提供数据或权限，授予对缓存块的访问权。也用于确认脏数据的主动写回。

通道E. 主端提供此次事务完成的最终确认消息，从端可用来事务序列化。

Messages

在五个通道内，TL-C具体说明了组成三种操作的十个消息。

消息	操作码	操作	A	B	C	D	E	响应
Acquire	6	Acquire	y	.	.	.	.	Grant, GrantData
Grant	4	Acquire	.	.	.	y	.	GrantAck
GrantData	5	Acquire	.	.	.	y	.	GrantAck
GrantAck	-	Acquire	.	.	.	.	y	
Probe	6	Probe	.	y	.	.	.	ProbeAck, ProbeAckData

消息	操作码	操作	A	B	C	D	E	响应
ProbeAck	4	Probe	.	.	y	.	.	
ProbeAckData	5	Probe	.	.	y	.	.	
Release	6	Release	.	.	y	.	.	ReleaseAck
ReleaseData	7	Release	.	.	y	.	.	ReleaseAck
ReleaseAck	6	Release	.	.	.	y	.	

表8.2 权限转换操作消息总览

Permissions Transitions

逻辑上，Transfer是对权限进行操作，因此组成其的消息必须指定预期的结果：升级到更高权限，降级到更低权限，或者一个保持权限不变的无操作。这些变化是根据它们对特定地址的一致性树形状的影响来指定的。我们将可能的权限转换组合分解为六个子集；可以使用不同的子集作为特定消息的参数，在下一小节中定义。

类别	内容
Permissions	None, Branch, Trunk
Cap	toT, toB, toN
Grow	NtoB, NtoT, BtoT
Prune	TtoB, TtoN, BtoN
Report	TtoT, BtoB, NtoN

表8.3 权限转换的分类

表8.3展示了基于TileLink的一致性协议所需的权限转换。它们被分成四个子集。

Prune: 权限降级，缩小一致性树。相比过去，完成操作之后具有较低的权限。

Grow: 权限升级，增大一致性树。相比过去，完成操作之后具有较高的权限。

Report: 包含权限保持不变，但报告当前权限状态。

Cap: 包含权限更改，但不指定原始权限是什么，而只指定它们应该成为什么。

数据流与波形

.传输操作引入新的事务流，这些事务流可以组成完整的缓存一致性协议事务。图8.3概述了三个新数据流。‘Acquire’请求总是触发递归的’ Release’请求和响应。根据块的权限的状态和一致性协议的不同，一个请求可能触发一个或多个递归的 release 或 probe 操作。

图8.3展示了一个消息流，它详细地描述了一个包含所有三个新操作的事务。在此流程中，一个主端通过acquiring目标数据块本地副本的读取或写入权限，来响应访存操作请求。在这个事务完成之后，主端已经获得了对缓存块的读写权限，以及该块数据的副本。其他的主端被probe，以迫使他们释放他们对该块的权限，并写回他们所拥有的脏数据。此外，发射’ Acquire ‘的主端也可以使用一个’ Release ’来主动释放他们对缓存块的权限。通常，当缓存必须清除包含脏数据的块，以便用重新填充到缓存中的块替换它时，就会触发这种类型的事务。在此事务完成后，主端失去了读取或写入该缓存块或该数据副本的权限。如果从端能够使用目录追踪到哪个主端拥有块的副本，那么这个元数据(metadata，附属信息)要被更新，以反映两个块的权限变化。

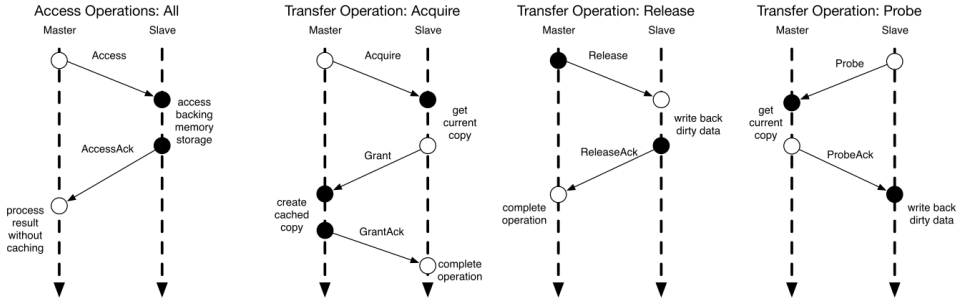

图 8.2: TileLink操作的事务流概图，黑点的移动说明事务的序列化被该操作影响

1. 缓存主代理向从代理发送Acquire。
2. 为保证给响应消息留足够空间，主代理主动发送Release，即写回操作。
3. 从代理访问存储结构，以完成写操作。
4. 从代理通过ReleaseAck确认写回操作已完成。
5. 从代理向其他主代理发送必需的Probe。
6. 从代理等待所有被Probe的主代理返回ProbeAck。
7. 若有需要，从代理还需访问存储结构。
8. 从代理向原请求设备发送Grant。
9. 原主代理以GrantAck说明此次事务成功完成。

虽然这三个流程构成了所有涉及缓存块传输的TileLink事务的基础，但是当它们临时地重叠或分层地组合时，会出现一些边界情况。现在我们将讨论TileLink如何管理并发性，并分散到各主从代理之间。

TileLink假设消息并不完全是点到点的有序的传递。事实上，来自高优先级通道的消息必须能够在网络中先于较低优先级的消息处理，即使它们的目标是相同的代理。从端充当连接到它的所有主端的一个同步节点。由于每个事务都必须通过发送给从端的一条 Acquire 消息作为初始消息，因此从端可以轻松地对事务进行排序。一个非常安全的实现方式是一次只接受一个事务，但是这种方式对性能影响巨大，而且事实证明我们可以在继续提供正确的序列化的同时增加并发性。尽管事务有着天然的分布式属性，对代理的行为施加一些限制，仍能够保证事务间的有序性。

对TileLink代理上的并发限制在发射或阻塞请求消息方面最容易理解。所有请求消息都会引发响应消息，并且响应消息保证最终产生forward progress。但是，在某些情况下，在收到还未处理的响应消息之前，不应该发出针对同一块的递归请求消息。我们按照请求消息类型来对这些情况分类：

Acquire: 如果在块上有一个悬而未决的Grant块，主端就不应发射一个Acquire。一旦Acquire被发出，主端不应该在该块发出进一步的Acquire，直到它收到一个Grant。

Grant: 在块上有一个悬而不决的ProbeAck时，从端不应该发射Grant。一旦发出了一个Grant时，从端不应该在该块上发射Probe，直到它收到一个GrantAck。

Release: 在块上有一个悬而不决的Grant时，主端不应该发射一个Release。一旦发出了一个Release后，主端不能发射ProbeAck，直到它收到来自从端确认写回操作完成的ReleaseAck。

Probe: 在块上有一个悬而不决的GrantAck时，从端不应该发射一个Probe。一旦发射了一个Probe,从端不能进一步发射Probe，直到它收到一个ProbeAck。

1. 主代理A先发送Acquire，但由于网络延迟，后到从代理。
2. 主代理B后发送Acquire，但先到达从代理，被序列化在A的前面。

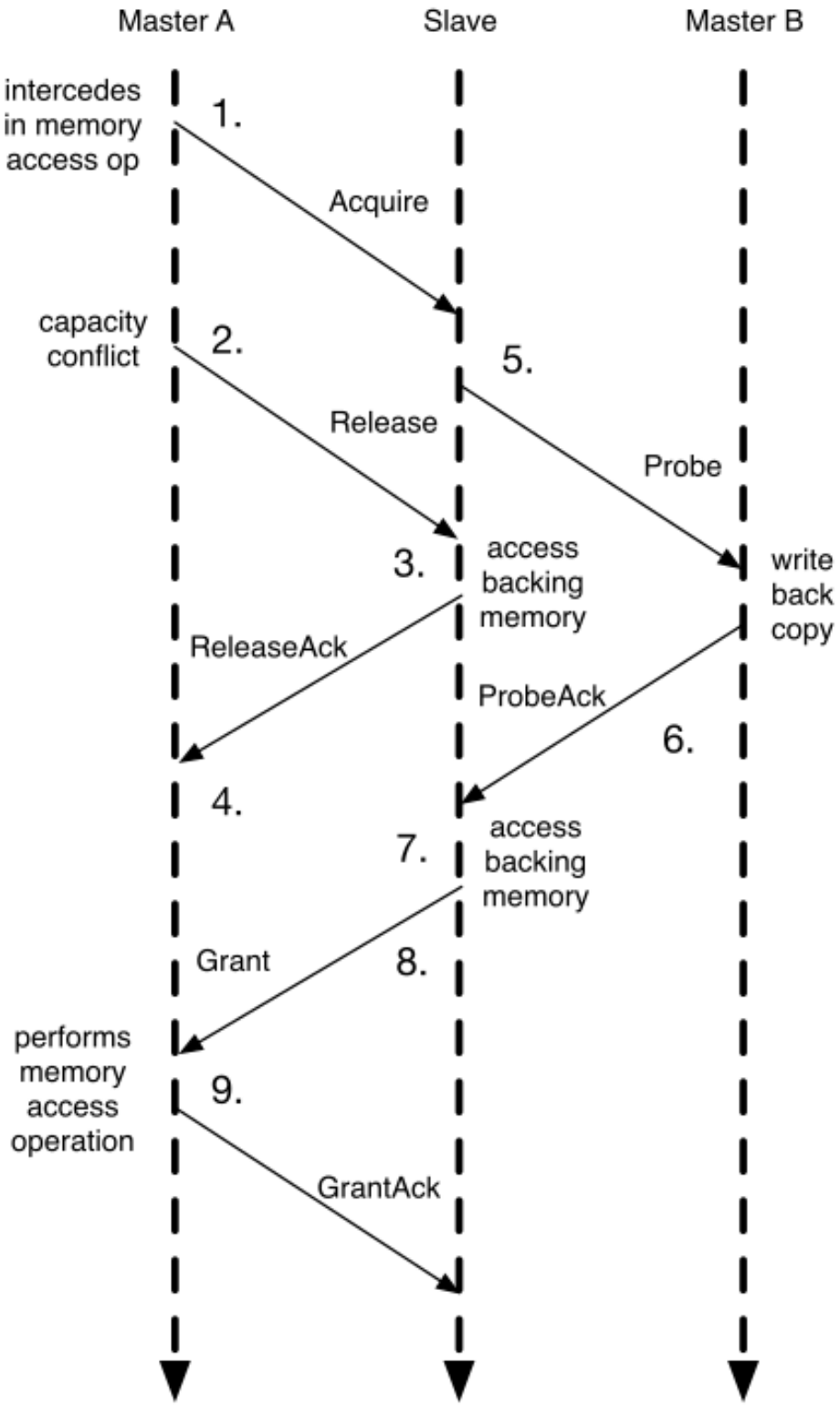

图 8.3: 三种转换操作事务流概览

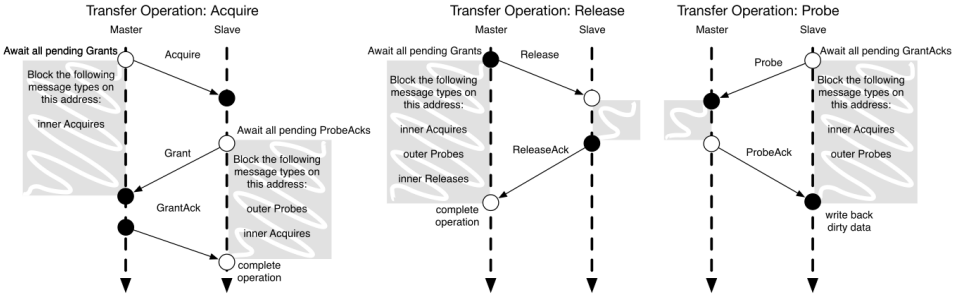

图 8.4: 转换操作的并发管理规则，是在4.2.2节描述转发推进规则(forward progress rule)的扩展

3. 从代理向A发送Probe，即使A还在等待Grant，也必须先处理Probe。
4. 从代理接收到A的ProbeAck后，向B发送Grant。
5. 从代理接收到A的Acquire，但由于正等待B的GrantAck，所以现在还不能处理这个请求。
6. 一旦接收到B的GrantAck，A的事务就可以正常处理了。
7. 从代理向B发送Probe，但这个操作被在上一个Grant之后。
8. 从代理向A发送合适类型的Grant(包括数据副本)，说明A在Acquire后被Probe过。

1. 主代理A向从代理发送Acquire。
2. 与此同时，主代理B通过Release主动剔除相同的数据缓存块。
3. 从代理向B发送Probe。
4. 从代理等待每个发送出的Probe，但可以处理主动发起的Release。从代理发送ReleaseAck确认主动写回的操作完成。
5. B在接收到写回确认前不处理Probe。
6. 在从代理接收到B的ProbeAck后，A的事务就可以正常执行了。

我们现在提供了一些示例流程，演示了在以消息序列图格式中所遵守的并发限制。图8.5展示了延迟Probe请求的场景。即使网络中有一个悬而未决的Grant，主端必须继续处理和响应Probe。从端必须在响应Acquire升级权限的Grant中包含数据的最新副本，除非他们确定自Acquire发射以来，没有对主端进行过Probe。假设从端在处理获取同一数据块的第二个事务时阻塞，那么关键问题就变成了:从端在什么时候处理一个挂起的Acquire是安全的?如果我们假设点对点有序地将消息传递给特定的代理，那么对于从端来说，仅仅将Grant消息发送给初始的主端源就足够了。从端可以在块上处理更多的事务，然后会依次向同一主服务器发送更多的Probe和Grant。由于此顺序没有得到保证，因此我们转而依赖GrantAck消息来允许从端序列化这两个事务。

现在我们来看第二个关于并发限制的例子，这个例子被放在了主端上。如果一个主端在一个块上有一个未完成的Release事务，那么它就不能用ProbeAcks响应该块上的一个传入的Probe请求，直到它从从端收到一个确认写回完成的ReleaseAck。图8.6以消息序列图的形式展示了这个场景。这个限制序列化了主动回写相对于正在进行的产生Probes的Acquire操作的顺序。在Acquire事务完成之前，从端不能简单地阻塞主动释放的Release事务，因为事务中的ProbeAck消息可能被阻塞在自动Release后的网络中。从从代理的角度来看，它必须处理一个情况，其正在接受一个块的主动Release时，一个主端正试图Acquire该块。从端必须接受主动的Release，以及已经发送的Probe消息所产生的任何ProbeAck，然后在他们的事务被认为完成之前向每个主端提供Release和Grant消息。主动写的数据可以与Grant来响应原始请求者，但事务在从端收集到预期数量的ProbeAcks之前无法完成。这个场景是两个事务消息流被从代理合并的示例。

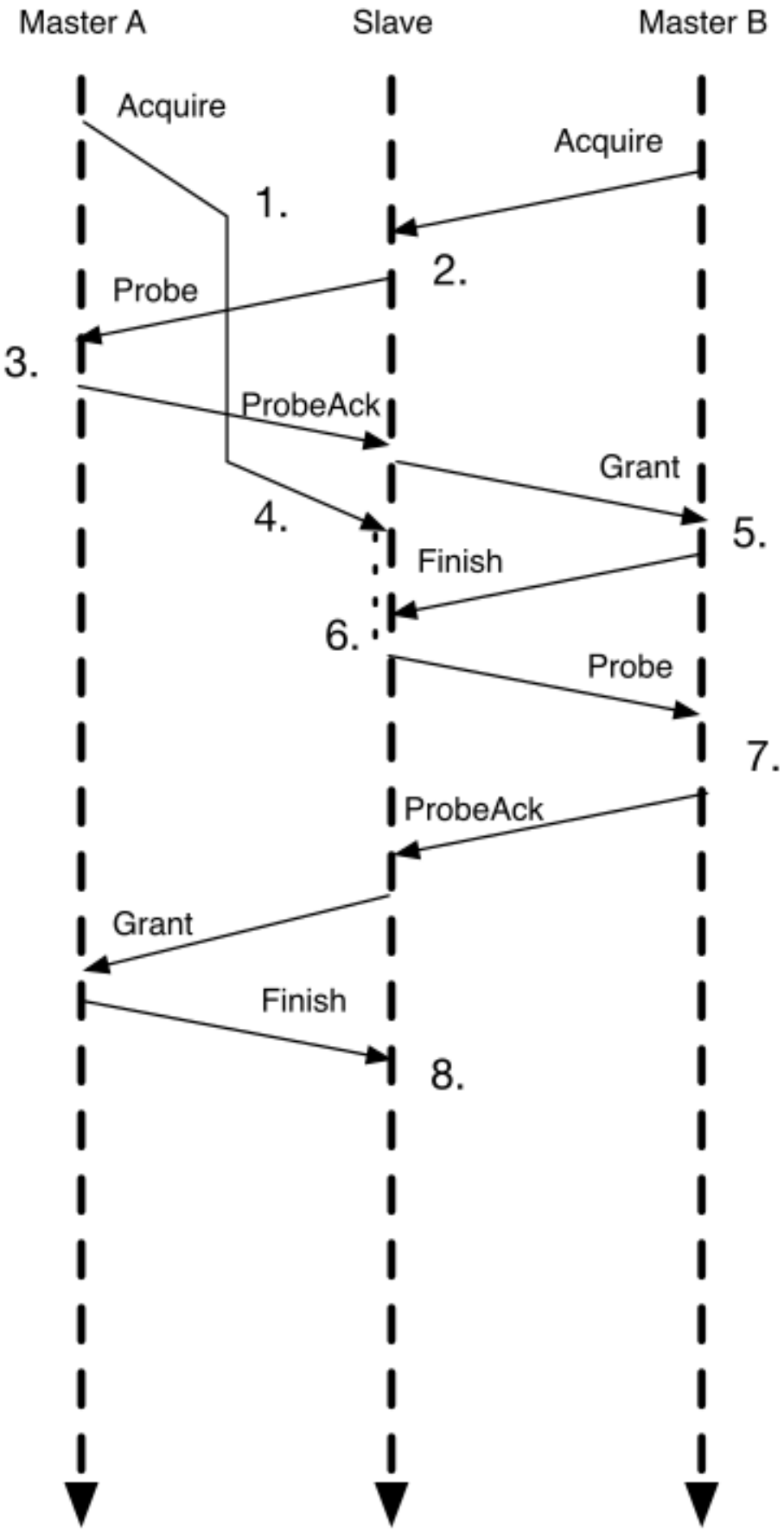

图 8.5: 在从代理端通过GrantAck来序列化相互交织的消息流的Grant和Probe

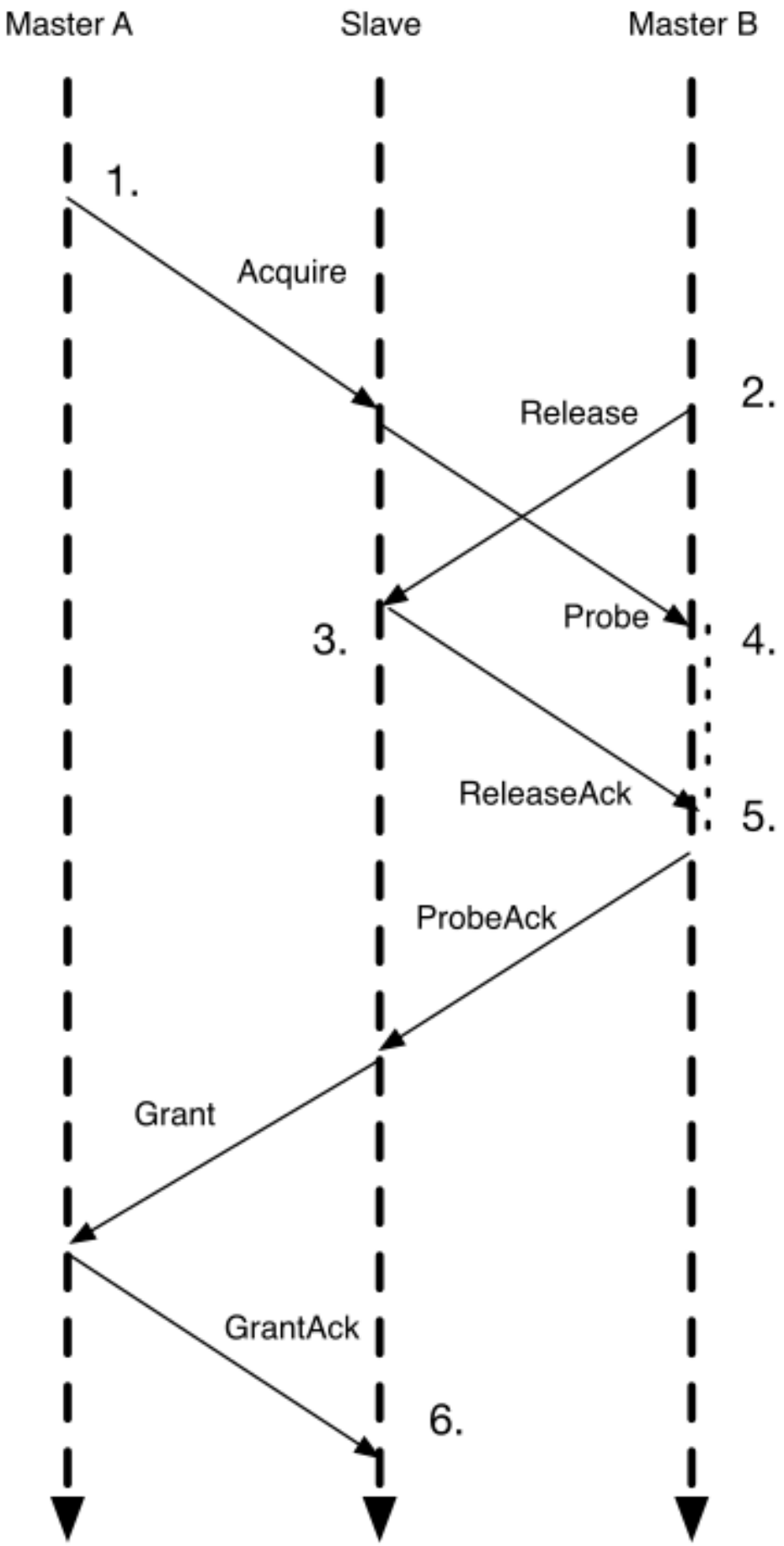

图 8.6: 从代理端序列化相互交织的消息流的Release和Probe

主代理上的最后一个并发限制是，仅当事务可以通过惟一的事务标识符彼此区分时，才可以对同一个块发出多个通道A请求。例如，主代理缓存在读未命中下又写未命中，可能会在提供读权限的Grant到达之前发出请求写权限的Acquire请求。但是，它必须为第二个“Acquire”使用唯一的事务ID，即使它的目标是相同的地址。主代理不能期望从代理以任何特定的顺序序列化多个未完成的Acquire，并且它必须为它所接收到的第一个Grant[Data]发送一个GrantAck，而不需要等待第二个Grant[Data]

TL-C Messages

权限传输新添的三个通道有六个新消息，另又新添了一个通道A消息、三个通道D消息。新的通道是B、C和E，新的消息类型是Acquire、Probe、ProbeAck[Data]、Release[Data]、ReleaseAck、Grant[Data]和GrantAck。

Acquire

Acquire消息是主代理计划在本地缓存数据块的副本时，发起的请求消息类型。主代理还可以使用这种消息类型来升级他们已缓存块上的权限(例如，获得只读副本的写权限)。与Get消息一样，Acquire消息本身不包含数据。表8.4显示了用于此消息类型的通道A字段的编码。

a_opcode 必须为Acquire, 编码值为6.

a_param 指明了主代理想要进行的权限改变的具体类型。从表8.3的类别中选择可能的转换

a_size 指明了主端想要缓存的数据的总数，以log2(bytes)的形式表示。

a_address必须与a_size对齐.

a_mask提供字节选择通道，在本例中指示读取哪个字节。有关详细信息，请参阅第4.6节。a_size、a_address和a_mask相互对应。Acquire必须具有自然对齐的连续mask。

a_source 发出此请求的主代理的ID，在从代理响应时，可以作为目标索引，确保回复消息被正确的路由。

通道A	信号类型	信号位宽	信号内容
a_opcode	C	3	必须为 Acquire (6)
a_param	C	3	权限转换: Grow (NtoB, NtoT, BtoT)
a_size	C	s	2^n 个字节会被从端读取并返回
a_source	C	c	发射该请求的主端源标识符
a_address	C	a	转换操作的目标地址
a_mask	D	w	要读取的字节通路
a_data	D	8w	忽略，可以是任何值

表8.4: 通道A的消息的字段

Probe

Probe消息是从代理用来查询或修改由特定主代理存储的数据块的缓存副本的权限的请求消息。从代理由于响应另一个主代理的Acquire，或是是主动发起，可以废除主代理对一块缓存块的权限。表8.5显示了该消息类型的通道B的所有字段。

b_opcode必须为Probe, 编码为6。

b_param指示了从代理想要进行的权限更改的具体类型。从表8.3的Cap类别中选择可能的转换。探查主代理，将其权限覆盖在比他们已有的权限更为宽松的级别是允许的，并且不会导致权限的更改。

b_size指示了请求代理希望探查的数据的总量，以log2(字节)的形式。如果为了响应这个探查而回写脏数据，b_size表示生成的ProbeAckData消息的大小，而不是这个特定的probe消息。

b_address 必须与b_size对齐

b_mask 提供字节选择通道，指示要探查哪些字节。有关详细信息，请参阅第4.6节。

b_address与b_mask需要相互对应。 Probe消息必须有连续的mask。

b_source是此请求的目标主代理的ID。它用于将请求传输到特定的缓存。详情见第5.4节。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为Probe (6)
b_param	C	3	权限转换: Cap (toN, toB, toT).
b_size	C	s	2^n 字节会被主端探查，并且可能返回
b_source	C	c	该请求的目标主端源标识符
b_address	C	a	转换操作的目标地址，以字节的形式
b_mask	D	w	要读取的字节通路
b_data	D	8w	忽略，可以是任何值

表8.5: 通道B的消息的字段

ProbeAck

ProbeAck消息是主代理用来回复Probe的消息。表8.6显示了该消息类型的通道C的所有字段。

c_opcode必须为ProbeAck,编码为4。 c_param .指示了主代理中由于Probe而发生的权限更改的具体类型。从表8.3的Shrink或Report类别中选择可能的转换。前者表示权限被降低，而后者则报告当前的权限是什么并继续保持不变。

c_size指示了以log2(bytes)形式表示的，被Probe的数据总量。这个消息本身不携带数据。

c_address用于将回复消息路由至初始的请求方，它必须与c_size对齐。

c_source 该回复的来源主代理的ID。

c_data 被忽略，可以取任何值。

c_error 保留，但必须置为0.

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为ProbeAck (4)
c_param	C	3	Permissions transfer: Shrink or Report (TtoB, TtoN, BtoN, TtoT, BtoB, NtoN)
c_size	C	s	2^n 字节会被探查;从b_size拷贝而来
c_source	C	c	回复消息的主端源标识符; 拷贝自b_source
c_address	C	a	转换操作的目标地址; 拷贝自 b_address
c_data	D	8w	被忽略，可以是任何值

通道C	信号类型	信号位宽	信号内容
c_error	F	1	保留; 必须为 0

表8.6: 通道C上的 ProbeAck 消息的字段

ProbeAckData

ProbeAckData消息是主代理使用的响应消息，用于确认接收到Probe，并写回发送请求的从代理所需的脏数据。表8.7显示了该消息类型的通道C的所有字段。

c_opcode 必须为 ProbeAckData, 编码为 5.

c_param 指示了主代理中由于Probe而发生的权限更改的具体类型。从表8.3的Shrink或Report类别中选择可能的转换。前者表示权限被降低，而后者则报告当前权限是什么并继续保持不变。

c_size 指示了以log2(bytes)的形式表示的被探测的数据总量，以及此消息中包含的数据量。

c_address用来将响应消息路由到初始的请求者处。

c_source该响应消息的源头的主端的ID, 拷贝自 b_source.

c_data 包含被此消息访问的数据。在ProbeAckData的各拍之间能被改变的数据称为burst。

c_error 指示了主端进行访存操作时发生了一个错误。 在一个burst的最后一拍，错误标志可以被拉高。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为ProbeAckData (5)
c_param	C	3	权限转换: Shrink or Report (TtoB, TtoN, BtoN, TtoT, BtoB, NtoN)
c_size	C	s	2^n 字节被探查并被写回; 拷贝自 b_size
c_source	C	c	该响应消息的主端源标识符; 拷贝自 b_source
c_address	C	a	转换操作的目标地址; 拷贝自b_address
c_data	D	8w	数据载荷
c_error	F	1	主端无法完成请求服务

表8.7: Channel C上的消息的字段

Grant

Grant消息是一个响应也是一个请求消息，从代理使用它来确认接收到一个Acquire，并提供访问缓存块的权限给原始发送请求的主代理。表8.8显示了用于此消息类型的通道D字段的编码。

d_opcode必须为Grant,编码为4.

d_param指示由于Acquire请求消息，从代理允许在主代理的块的缓存副本上发生的访问的具体类型。从表8.3的Cap类别中选择可能的权限转换。增加权限而不说明初始权限。权限可能超过原始请求消息的a_param字段中所请求的。

d_size包含正在转换权限的数据的大小，尽管此特定消息本身不包含数据。必须与原始的a_size相同。

d_sink是发出此消息的代理，用于该消息的通道E响应消息的索引，而在原始通道A请求中，d_source应该从a_source中保存下来，现在被重新使用来将此响应路由到正确的目的地。详情见第5.4节。

d_data被忽略，并可以指定为任意值。

d_error被保留并且必须为0.

通道D	信号类型	信号位宽	信号内容
d_opcode	C	3	必须为 Grant (4)
d_param	C	2	权限转换: Cap (toT, toB, toN)
d_size	C	s	2^n个字节被从端访问; 拷贝自a_size
d_source	C	c	接收该相应的主端源标识符; 拷贝自a_source
d_sink	C	m	发射该请求的从端sink标识符
d_data	D	8w	被忽略，可以是任何值
d_error	F	1	保留必须为0

表8.8: 通道D上的Grant消息的字段

GrantData

GrantData消息既是响应也是请求消息，从代理使用它向原始请求主代理提供确认消息以及数据块副本。表8.9显示了用于此消息类型的通道D字段的编码。

d_opcode必须为GrantData,编码为5.

d_param 作为Acquire请求的结果，指示了从代理允许在主代理中缓存副本上进行的具体访问类型，从表8.3的Cap类别中选择可能的权限转换。增加权限而不指定原始权限。权限可能超过原始请求的a_param字段中所请求的权限。

d_size 包含正在转换权限的数据块的大小，该大小对应于使用此特定消息发送的数据的大小。必须与原始的a_size相同。

d_sink 是发出此响应消息的代理的标识符，用于路由其通道E的响应，而在原始通道A请求中，a_source中保存下来，现在被重新使用以将此响应路由到正确的目的地。详情见第5.4节。

d_data包含被此操作传输的数据，此数据会被主代理所缓存。

d_error知识了当从端试图进行转换操作时发生了一个错误。

通道D	信号类型	信号位宽	信号内容
d_opcode	C	3	必须为 GrantData (5)
d_param	C	2	权限转换: Cap (toT, toB, toN)
d_size	C	s	2^n 被从端传输的字节数; 拷贝自a_size
d_source	C	c	接收该请求的主端源标识符; 拷贝自a_source
d_sink	C	m	发射相应消息的从端sink标识符
d_data	D	8w	数据载荷
d_error	F	1	从端无法服务该请求

表8.9: Channel D上消息的字段

GrantAck

GrantAck响应消息被主代理用来提供事务完成的最终确认消息，同时也被从代理用来确保操作的全局序列化。表8.10显示了通道E上此消息的所有字段。

e_sink应该被从前面的Grant[Data]消息中的d_sink中保存，现在正在被重用，以将此响应消息路由到正确的目的地。

通道E	信号类型	信号位宽	信号内容
e_sink	C	m	接收响应消息的从端sink标识符; 拷贝自d_sink

表8.10:GrantAck消息的字段

Release

Release消息是主代理用来主动降低其对一个缓存数据块的权限的请求消息。表8.11显示了该消息类型的通道C的所有字段。

c_opcode必须为Release,编码为6.

c_param指示了主代理发起的权限更改的具体类型。可能的转换是从表8.3的Shrink类别中选择的，该类别指示了权限是什么以及它们正在变成什么。

c_size 以log2(bytes)的形式表示权限将被Release的缓存数据总量。这个消息本身不携带数据。

c_address用于路由响应消息去往管理该地址的从代理，必须与c_size对齐。

c_source是作为该请求的源的主代理的ID。ID不必与最初用于Acquire该块的ID相同，尽管它必须对应于相同的主代理。

c_data被忽略，可以取任何值。

c_error 保留，并且必须置为0。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为Release (5)
c_param	C	3	权限转换: Shrink or Report (TtoB, TtoN, BtoN, TtoT, BtoB, NtoN)
c_size	C	s	2^n 个字节被主端降级
c_source	C	c	该请求的主端源标识符
c_address	C	a	转换操作的目标地址，以字节形式
c_data	D	8w	忽略，可以是任何值
c_error	F	1	保留，必须置为0

表8.11: Channel C上消息的字段

ReleaseData

ReleaseData消息是主代理发起的请求消息，用于主动降低对一块缓存数据块的权限。并将脏数据写回管。表8.12显示了该消息类型的通道C的所有字段。

c_opcode 必须为ReleaseData,编码为6.

c_param指示主代理发起的权限更改的具体类型。可能的转换是从表8.3的Shrink类别中选择的，该类别指示了权限是什么，以及它们将变成什么。

c_size 指示了以log2(bytes)的形式表示的释放权限的缓存数据的大小，以及此消息中包含的数据大小。

c_address用于路由响应消息去往原始的请求者出，必须与c_size对齐.

c_source该响应消息发起者主代理的ID

c_data包含操作回写的脏数据。数据可以在一个burst的ReleaseData的各拍间进行改变。

c_error 指示了试图处理访存操作时发生了一个错误。此标志位可用于指示从缓存中清除数据的内存污染。错误标志应该在burst的最后一拍被拉高。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为ReleaseData (6)
c_param	C	3	权限转换: Shrink (TtoB, TtoN, BtoN, TtoT, BtoB, NtoN)
c_size	C	s	2^n 个字节被主端写回
c_source	C	c	该响应消息的主端源标识符
c_address	C	a	转换操作的目标地址, 以字节形式
c_data	D	8w	数据载荷
c_error	F	1	主端无法写回数据

表8.12: Channel C上的ReleaseData消息的字段

ReleaseAck

ReleaseAck消息是一个从代理发起的响应消息，用来响应“Release[Data]”，它反过来用于确保从代理的操作的全局序列化。表8.13显示了用于此消息类型的通道D字段的编码。

GrantAck响应消息被主端用来提供事务完成的最终确认，表8.10显示了该消息在通道E上的所有字段。

d_opcode必须为ReleaseAck,编码为 6.

d_param 被保留并且必须为0.

d_size包含权限被转换的数据的大小，尽管此特定消息本身不包含数据。d_size可以从Release[Data]消息中的c_size中获得。

d_source应该从前面的Release[Data]消息中的c_source中获得，现在被重用，以将响应信息路由到正确的目的地。d_sink被忽略，并且不需要在Release的过程中保持唯一。详情见第5.4节。

d_data被忽略，并且可以取任何值。

d_error 指示了从端在进行内存访问操作时出现了一个错误。

通道D	信号类型	信号位宽	信号内容
d_opcode	C	3	必须为 ReleaseAck (6)
d_param	C	2	保留;必须为0
d_size	C	s	被传输的字节; 拷贝自c_size
d_source	C	c	接收该响应的主端源标识符; 拷贝自c_source
d_sink	C	m	忽略，可以取任何值
d_data	D	8w	忽略，可以取任何值

通道D	信号类型	信号位宽	信号内容
d_error	F	1	从端无法服务该请求消息

表8.13: Channel D 上消息的字段

TL-UL and TL-UH Messages on Channel B and Channel C

除了三个新操作(Acquire、Probe、Release)之外，TL-C重新定义了所有在通道B和C上的TL-UH的操作，这允许那些通道被用来转发Access和Hint操作给远端的缓存数据所有者。换句话说，其实现可以选择使用基于更新的协议，而不是基于失效的协议。

Get

Get消息是一个代理发出的请求，目的是读取一个特定的数据块。表8.14显示了用于此消息类型的B通道字段的编码。

b_opcode 必须为Get, 编码为4.

b_param 现在为未来的Hint操作保留，置为0.

b_size 指示了以log2(bytes)的形式表示请求代理希望读取的数据总量。b_size表示生成的AccessAckData消息的大小，而不是当前的Get消息。

b_address必须与b_size对齐.

b_mask 提供字节选择通道，在本例中指示读取哪些字节。有关详细信息，请参阅第4.6节。b_size、b_address和b_mask需要相互对应。Get消息必须具有连续的mask。

b_source 是此请求的目标的主代理的ID。它用于路由请求。详见第5.4章。

b_data 被忽略并且可以取任何值。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为Get (4)
b_param	C	3	保留必须为0
b_size	C	s	2^n 被主端读取并返回的字节
b_source	C	c	该请求的目标主端源标识符
b_address	C	a	访问的目标地址,字节形式
b_mask	D	w	要读取的字节通路
b_data	D	8w	被忽略，可以是任何值

表8.14: Channel B 上Get消息的字段

PutFullData

PutFullData消息是一个代理的请求，目的是写特定的数据块。表8.15显示了用于此消息类型的通道B字段的编码。

b_opcode必须为 PutFullData, 编码为 0.

b_param 现在为未来的hints保留，必须为0

b_size 以log2(bytes)的形式指示了请求代理想要写的数据总量。在本例中b_size代表了该请求消息的大小。

b_address 必须与b_size对齐. 从 b_address到b_address+2**b_size-1的整个内容都会被写。

b_mask 提供了字节选择通路，在本例中指示写入哪些字节。有关详细信息，请参阅第4.6节。b_mask的一个位对应于写入的一个字节数据。b_size、b_address和*_mask需要相互对应。PutFullData必须有一个连续的mask，如果b_size大于或等于物理数据总线，那么所有b_mask都必须为高。

b_source 是该请求的目标从代理的ID。它用于路由请求消息。

b_data为被实际写入的数据载荷

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为PutFull (0)
b_param	C	3	保留必须为0
b_size	C	s	2^n 字节会被主端写入
b_source	C	c	该请求的目标主端源标识符
b_address	C	a	访存目标地址, 字节形式
b_mask	D	w	被写的字节通路, 必须为连续的
b_data	D	8w	被写入的数据载荷

表8.15: Channel B上的PutFullData的字段

PutPartialData

PutPartialData消息是一个代理的请求，目的是写特定的数据块。PutPartialData可用于在字节粒度上编写任意对齐的数据。表8.16显示了用于此消息类型的通道B字段的编码。

b_opcode必须为PutPartialData, 编码为1.

b_param 现在为未来的hints保留，并且必须为0.

b_size以log2(bytes)的形式指示了请求代理想要写的数据总量。b_size也表示此请求消息数据的大小。

b_address必须与b_size对齐，b_address到b_address+2**b_size-1 的部分字节会被写。

b_mask 提供字节选择通道，在本例中指示写入哪些字节。有关详细信息，请参阅第4.6节。b_mask的一位对应于写入的数据的一个字节。b_size、b_address和b_mask需要相互对应，但是PutPartialData写的数据可能少于b_size，这取决于b_mask的内容。b_mask的任何位的集合都必须包含在b_size的对齐区域中。

b_source 是该请求的目标的主接口的ID。它用于路由请求消息。

b_data是实际被写入的数据载荷。未被mask的一个字节的b_data都被忽略，可以取任何值。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为PutFull (1)
b_param	C	3	保留; 必须为 0
b_size	C	s	最多会有2^n字节被主端写入
b_source	C	c	该请求消息的目标主端源标识符
b_address	C	a	访存操作的目标基地址，以字节的形式
b_mask	D	w	要被写入的字节通路
b_data	D	8w	被写入的数据载荷

表8.16: Channel B上的 PutPartialData 的字段

AccessAck

AccessAck 向原始请求代理提供不带数据的确认消息。表8.17显示了用于此消息类型的通道C字段的编码。

c_opcode 必须为 AccessAck,编码为 0。

c_param 保留为TL-C操作码使用，并且应该指定为0。

c_size 包含被访问的数据的大小，尽管此特定消息本身不包含数据。大小和地址字段必须对齐。

c_address 必须与触发此消息的源请求消息的b_address一致。

c_source 是发出此响应消息的代理的ID。详见第5.4章。

c_data 被忽略，可以取任何值。

c_error 指示当主端尝试处理访存操作时发生错误。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为AccessAck (0)
c_param	C	3	保留，必须为0
c_size	C	s	2^n 字节会被主端访问
c_source	C	c	发送响应消息的主端的源标识符
c_address	C	a	操作的目标地址，以字节形式
c_data	D	8w	忽略，可以取任何值
c_error	F	1	主端无法服务该该请求消息

表8.17: Channel C 上的AccessAck的字段

AccessAckData

AccessAckData 向原始请求代理提供带有数据的确认消息。表8.18显示了用于此消息类型的通道C字段的编码。

c_opcode 必须为 AccessAckData，编码为 1.

c_param 被保留为TL-C的操作码，并且必须指定为0。

c_size 包含被访问的数据的大小，对应于与此特定消息相关联的数据的大小。大小和地址字段必须对齐。

c_address 必须与触发此消息的源请求消息的b_address一致。

c_source 是发射该响应消息的代理的ID，详情可见5.4节。

c_data 包含操作访问的数据。数据可以在一个burst的AccessAckData的拍之间改变。

c_error 指示了当主端尝试处理访存操作时发生错误。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为 AccessAckData (1)
c_param	C	3	保留，必须为 0
c_size	C	s	2^n 个字节被主端访问
c_source	C	c	发出该响应消息的主端源标识符
c_address	C	a	访问的目标地址，以字节形式
c_data	D	8w	携带数据的消息的数据载荷
c_error	F	1	主端无法服务该请求

表8.18: 通道C上AccessAckData消息的字段

ArithmeticData

ArithmeticData 消息是一个代理发出的请求，该代理希望访问一个特定的数据块，并使用算术操作来读-修改-写(read-modify-write)。表8.19显示了用于此消息类型的通道B字段的编码。

b_opcode必须为ArithmeticData, 编码为2.

b_param 指定要执行的具体原子操作。表7.3列出了所支持的算术操作集。它由{ MIN, MAX, MINU, MAXU, ADD }组成，分别表示有符号和无符号整数最小值和最打值，以及整数加法。

b_size 是算术操作数大小，它反映了请求数据和AccessAckData响应的大小

b_address 必须与b_size对齐。

b_mask 提供字节选择通道，在本例中指示读-修改-写哪些字节。有关详细信息，请参阅第4.6节。b_mask的一个位对应于原子操作中使用的数据的一个字节。b_size、b_address和b_mask需要相互对应(比如：mask在对齐范围内连续地全置为高)。

b_source是该请求的目标主接口的ID，用于路由请求消息。

b_data 包含算数操作的一个源操作数(另一个在目标地址)。当某一个字节中的b_data为unmasked时，会被忽略，可为任意值。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为 ArithmeticData (3)
b_param	C	3	见表7.3
b_size	C	s	2^n 个字节会被主端读与写
b_source	C	c	该请求的目标主端的源标识符
b_address	C	a	访问的目标地址，以字节形式
b_mask	D	w	被读写的字节通路
b_data	D	8w	用作操作数的数据载荷

表8.19: 通道B的ArithmeticData消息的字段

LogicalData

LogicalData 是一个代理发出的请求，该代理希望访问一个特定的数据块，并使用逻辑操作来读-修改-写。表8.20显示了用于此消息类型的通道B通道字段的编码。

b_opcode必须为 LogicalData,编码为 2.

b_param 指定要执行的具体原子性操作。表7.5列出了所支持的逻辑操作集。它由{ XOR, OR, AND, SWAP }组成，表示按位逻辑上的异或，或，与和操作数的简单交换。

b_size 是操作数大小，它反映了此请求数据和AccessAckData响应的大小。

b_address 必须与b_size对齐，详情可见4.6节。

b_mask 提供字节选择通道，在本例中指示读-修改-写哪些字节。有关详细信息，可参阅第4.6节。b_mask的一位对应于原子性操作中使用的数据的一个字节。b_size、b_address和b_mask需要相互对应(比如：mask在对齐范围内连续地全置为高)。

b_source 是此请求消息的目标主接口的ID。它用于路由请求消息。

b_data 包含逻辑操作的一个源操作数(另一个在目标地址)。当某一个字节中的’ b_data ’为unmasked时，会被忽略，可为任意值。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为 LogicalData (3)

通道B	信号类型	信号位宽	信号内容
b_param	C	3	见表 7.5
b_size	C	s	2^n 个字节会被主端读写
b_source	C	c	该请求的目标从端的源标识符
b_address	C	a	访问的目标地址，以字节形式
b_mask	D	w	被读写的字节通路
b_data	D	8w	被写的数据载荷

表8.20: 通道B上的消息字段

Intent

Intent 是一个代理发出的请求消息，该代理想要表明将来打算访问一个特定的数据块。表8.21显示了用于此消息类型的通道B通道字段的编码。

b_opcode 必须为 Intent, 编码为 5。

b_param 指定了此Hint操作所传递的具体意图。应注意，它的效果作用于从接口和从接口所连接的所有更外层节点。表7.7列出了所支持的intention的集合。它由{ PrefetchRead, PrefetchWrite }组成，表示prefetch-data-with-intent-to-read和prefetch-data-with-intent-to-write。

b_size 该intention所需要的数据的大小

b_address 必须与b_size对齐.

b_mask 提供字节选择通道，在本例中指示想要用到的字节。有关详细信息，请参阅第4.6节。b_size、b_address和b_mask需要相互对应。

b_source该请求消息的目标主接口的ID，用于路由请求消息。

b_data 忽略，可取任意值。

通道B	信号类型	信号位宽	信号内容
b_opcode	C	3	必须为 Intent(5)
b_param	C	3	Intention编码;见表7.7
b_size	C	s	2^n 个字节与intention有关
b_source	C	c	该请求消息的目标主端的源标识符
b_address	C	a	目标缓存块的地址，以字节的形式
b_mask	D	w	Hint应用的字节通路
b_data	D	8w	忽略，可以取任何值

表8.21: 通道B上的消息字段

HintAck

HintAck 用作Hint提操作的响应消息。表8.22显示了用于此消息类型的通道C字段的编码。

c_opcode必须为 HintAck，编码为2.

c_param保留且必须为0。

c_size 包含所操作的数据的大小，尽管此特定消息本身不包含数据。c_address只需要与c_size对齐即可。

c_sink 发出此响应消息的代理的ID，而c_source应该从请求消息中获得，现在被重用，以将此响应路由到正确的目的地。详见第5.4节。

c_data 被忽略，且可取任意值。

c_error 指示了主端在进行访存操作时出现了一个错误。

通道C	信号类型	信号位宽	信号内容
c_opcode	C	3	必须为HintAck (2)
c_param	C	3	保留必须为0
c_size	C	s	2^n 字节与hint相关
c_source	C	c	发送该响应消息的主端源标识符
c_address	C	a	操作的目标地址，以字节的形式
c_data	D	8w	忽略，可以是任何值
c_error	F	1	主端无法服务该请求消息

表8.22: 通道C上HintAck消息的字段