CXL

CXL è un protocollo di comunicazione che consente alla CPU e ad altre periferiche di comunicare con una bassa latenza e con un bandwidth elevato. Questa tecnologia nasce per supportare sistemi eterogenei nell'esecuzione di workload HPC (AI, ML).

Si basa sulle connessioni elettriche di PCIe, a cui però aggiunge la coerenza tra le memorie. CXL 3.0 si basa su PCIe 6.0 e ciò consente di ottenere una **bandwidth massimo teorico di 256GB/s**, grazie al trasfer rate di 64GT/s su 16 linee.

Lo scopo è quello di realizzare a livello hardware un'astrazione che mostra al programmatore un sistema a memoria comune, quando in realtà c'è un sistema a scambio di messaggi; viene gestito in maniera trasparente.

Il problema che nasce da questo sistema è rendere coerente la memoria di tutto il sistema e quindi sia la memoria della CPU sia quella degli acceleratori.

Protocollo

CXL è articolato su 3 livelli:

- Transaction layer
- Link layer
- Physical layer

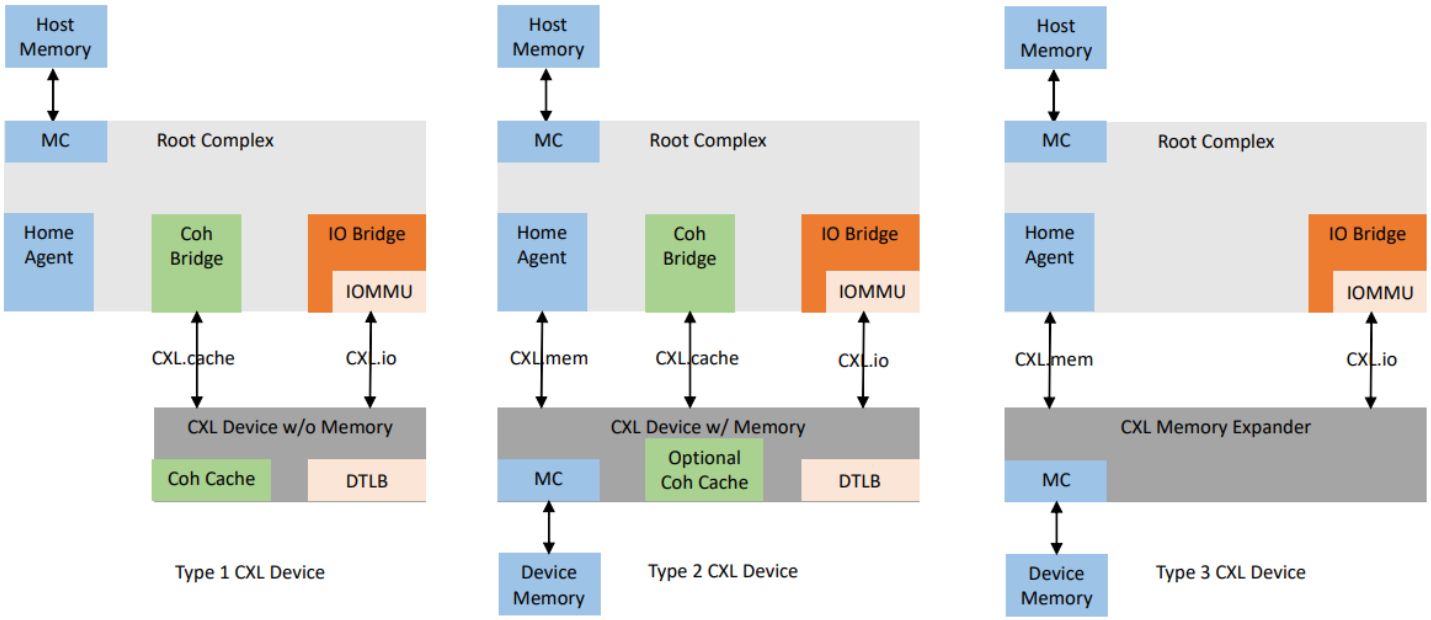
All'interno dei livelli operano i tre sotto-protocolli di CXL:

- CXL.io
- CXL.cache
- CXL.mem

Ognuno ha uno scopo diverso e viene usato in contesti diversi.

CXL prevede tre tipi di dispositivi:

- Tipo 1: acceleratori o dispositivi di caching, usano CXL.io e CXL.cache
- Tipo 2: acceleratori con un buffer di memoria privato, usano CXL.io, CXL.cache e CXL.mem
- Tipo 3: buffer di memoria, usano CXL.io e CXL.mem



CXL.io

È molto simile a PCIe e viene usato dall'host per: cercare, enumerare, configurare e gestire un qualsiasi dispositivo CXL.

CXL.memory

Il protocollo usato dall'host per accedere alla memoria, tramite semplici RD e WR.

CLX.cache

Questo protocollo viene usato dal device per avere un accesso coerente da device a CPU.

Componenti hardware

All'interno dei nodi host è necessario l'Home Agent, che è la componente hardware che si occupa di gestire la coerenza tra la cache. È trasparente, infatti i device non sanno della sua esistenza e di come funzioni.

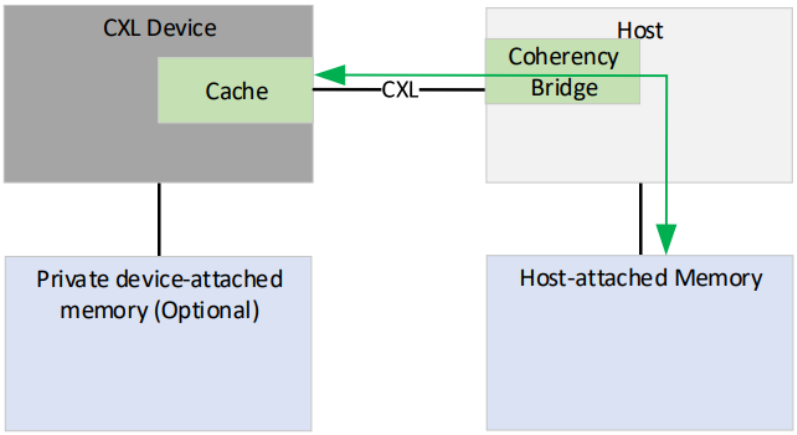
CXL device

I CXL device sono dispositivi non host collegati allo switch CXL, hanno la propria memoria dedicata esposta all'host si dice che sono *Host Managed Device Memory* (HDM), la coerenza può essere gestita in tre modi:

- Host-only coherent (HDM-H)
- Device coherent (HDM-D)
- Device coherent using Back-Invalidation Snoop (HDM-DB)

L'host e il device devono essere entrambi d'accordo sulla politica da usare.
Ogni host prevede una unità hardware chiamata *Home Agent* (HA) che si occupa di gestire la coerenza a livello di sistema per un dato indirizzo.

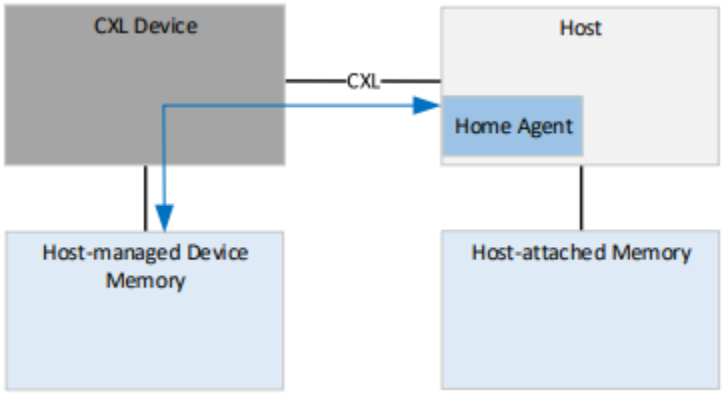
CXL type 1 device



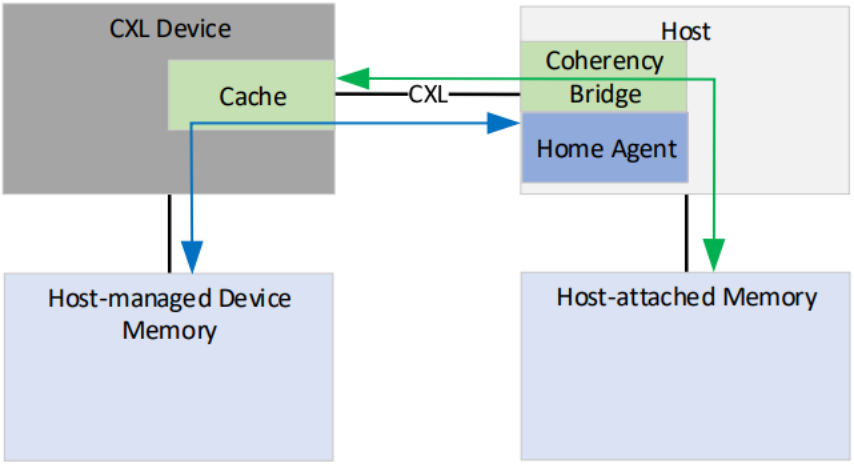
Questi dispositivi hanno una cache coerente con la memoria dell'host. Questa coerenza è garantita da CXL.cache, viene usato un meccanismo di Snooping standard, come quello usato dalla CPU.

CXL type 3 device

A questa categoria di dispositivi appartengono le espansioni di memoria, il dispositivo non fa richieste su CXL.cache, ma opera su CXL.memory per gestire le richieste dell'host. La gestione del dispositivo avviene tramite CXL.io.



CXL type 2 device



Questi dispositivi hanno sia una cache coerente che una memoria (DDR, HBM, ecc.) associata. La memoria del dispositivo viene sfruttata per la sua banda e l'host deve avere la possibilità di inserire ed estrarre dati dalla memoria del dispositivo senza costi in termini di overhead. In questo caso si tratta di HDM-D e HDM-DB.
Un esempio sono le GPGPU che hanno una memoria privata inaccessibile all'host e quindi deve esserci un modo per copiare con una banda elevata i dati dalla memoria dell'host a quella del dispositivo.
Ad alto livello ci sono due modi per risolvere la coerenza, il primo modo è detto **device coherent** e usa CXL.cache, si parla di **HDM-D**; il secondo modo usa il canale dedicato di CXL.mem chiamato Back Invalidation Snoop, in questo caso si tratta di **HDM-DB**.

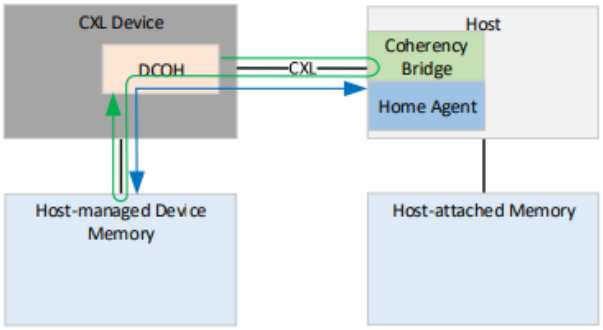
1ª soluzione: Bias-based coherency model

Questo modello definisce due stati di bias (pregiudizio, inclinazione): Host Bias e Device Bias.

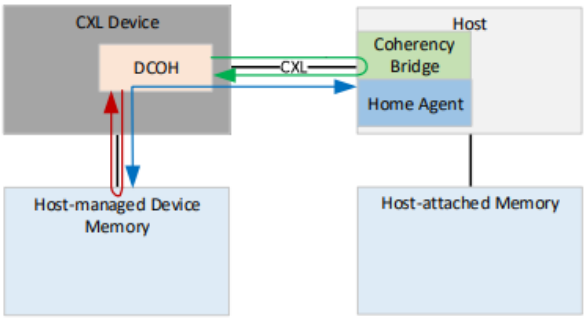
Quando la memoria del dispositivo è in stato **Host Bias**, al dispositivo appare come la memoria dell'host e se vi deve accedere manda la richiesta all'host che se ne occupa.
Quando invece la memoria del dispositivo è in stato **Device Bias**, il dispositivo è sicuro che l'host non ha il dato che cerca in cache, quindi vi accede senza mandare richieste all'host.
I vantaggi di questa soluzione sono:

- Aiuta a mantenere la coerenza per la memoria del dispositivo;
- Aiuta l'accesso alla memoria locale del dispositivo con una banda elevata perché non deve fare snoop;
- L'accesso alla memoria del dispositivo avviene in maniera coerente al modo in cui l'host accederebbe alla propria.

Host Bias:



Device Bias:



La gestione del bias mode viene fatta in due modi: SW assisted o HW.

2ª soluzione: Back-Invalidation Snoop Coherence for HDM-DB
CXL prevede un sistema di Global Observation (GO) per garantire ordinamento delle operazioni. CXL sfrutta il modello MESI per garantire coerenza tra le cache, che prevede quattro stati:

- **M**odified, il dato è presente in una sola cache, può essere letto o scritto, non è aggiornato in memoria;
- **E**xclusive, il dato è presente in una sola cache, può essere letto o scritto, è aggiornato in memoria;
- **S**hared, il dato è presente in più cache, può essere solo letto, è aggiornato con la memoria;
- **I**nvalid, il dato non è presente in cache.

Gestione delle cache tra peer

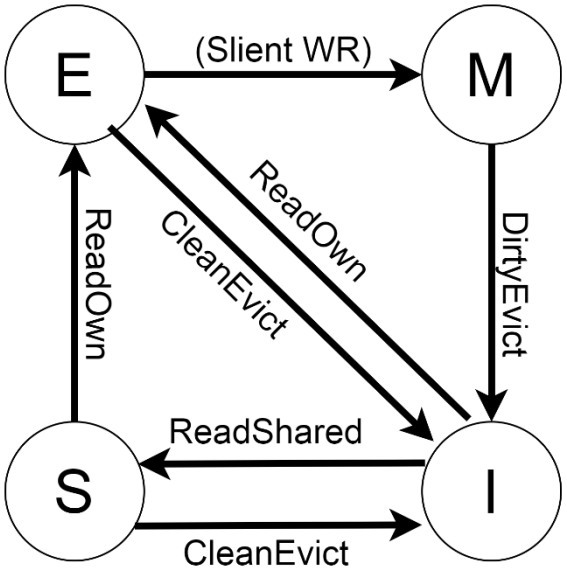
Tutte le cache dei peer sono gestite dall'HA, che sfrutta il concetto di snoop, questo consiste nel controllare lo stato della cache e causare modifiche nelle cache dei peer.

Gli snoop possono essere di diversi tipi:

- Snoop invalidate (SnpInv), fa passare la linea di cache in stato I e deve inviare tutti i dati modificati;
- Snoop data (SnpData), fa passare la linea di cache in stato S, deve inviare tutti i dati modificati;
- Snoop current (SnpCurr), non fa cambiare lo stato della cache, deve inviare i dati modificati.

MESI transitions in CXL

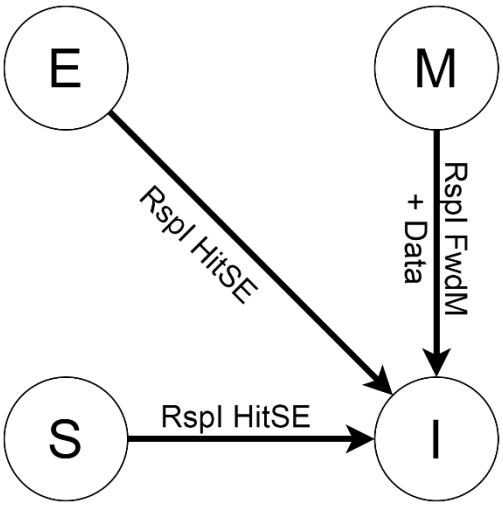
Richiesta da parte del device:



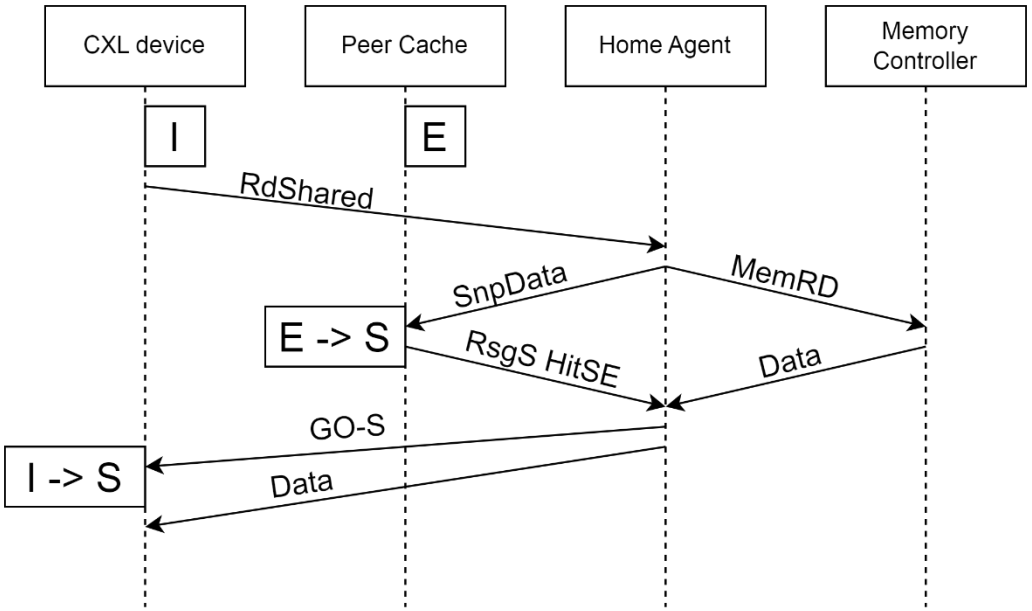
Il cambio di stato avviene sulla base dei messaggi inviati dal device. <Descrivere il comportamento>

Snoop Invalidate:

Questo è ciò che avviene nel device in seguito alla ricezione di una SnpInv da parte del HA. Le risposte inviate sono dirette all'HA dell'host.

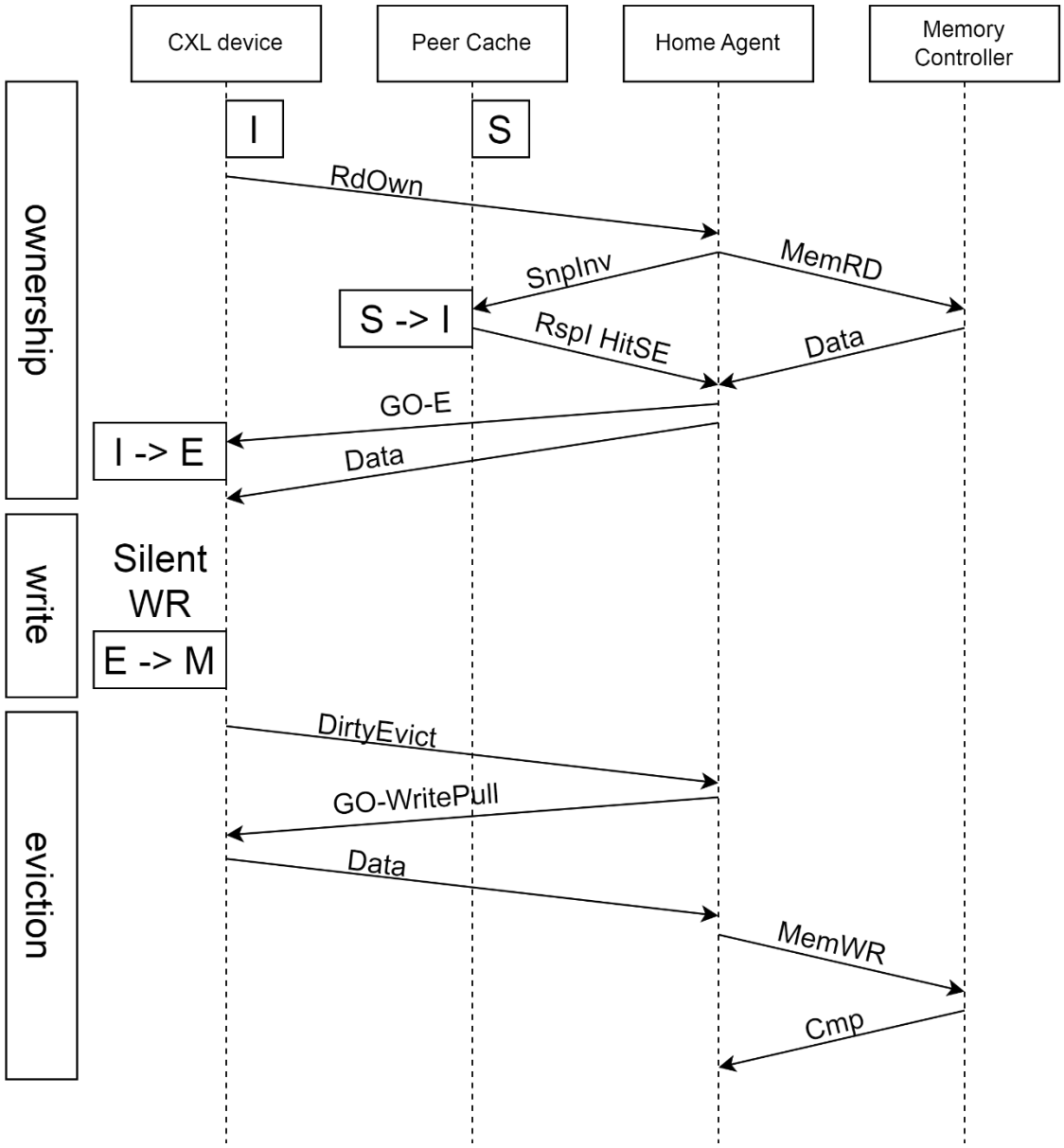


Flusso di RD del dispositivo CXL



RdShared legge una linea che viene salvata in stato S; **SnpData** risolve la coerenza con le cache dei peer; **RsgS HitSE** funge da ack per comunicare che la cache è passata in stato S; **GO-S** comunica di passare in stato S.

Flusso di scrittura del dispositivo CXL



Questo flusso può essere diviso in tre parti:

1. **Ownership**, in questa fase il dispositivo ottiene l'accesso esclusivo alla linea di cache;
2. **Write**, il dispositivo modifica il dato, si dice *silent* perché non viene notificato all'HA;
3. **Eviction**, al dispositivo non serve più il dato e quindi viene rimosso dalla cache.

Con **peer cache** si intende una qualsiasi cache all'interno del sistema, può essere quella di un dispositivo CXL, la cache locale alla CPU o cache remota di un'altra CPU. Sta all'HA capire dove sono queste cache e fare lo snoop.

Con **memory controller** si intende un qualsiasi memory controller, quello locale alla CPU, quello del device o quello di una CPU remota.

L'**Home Agent** può essere locale alla CPU, oppure remoto.

UCIe

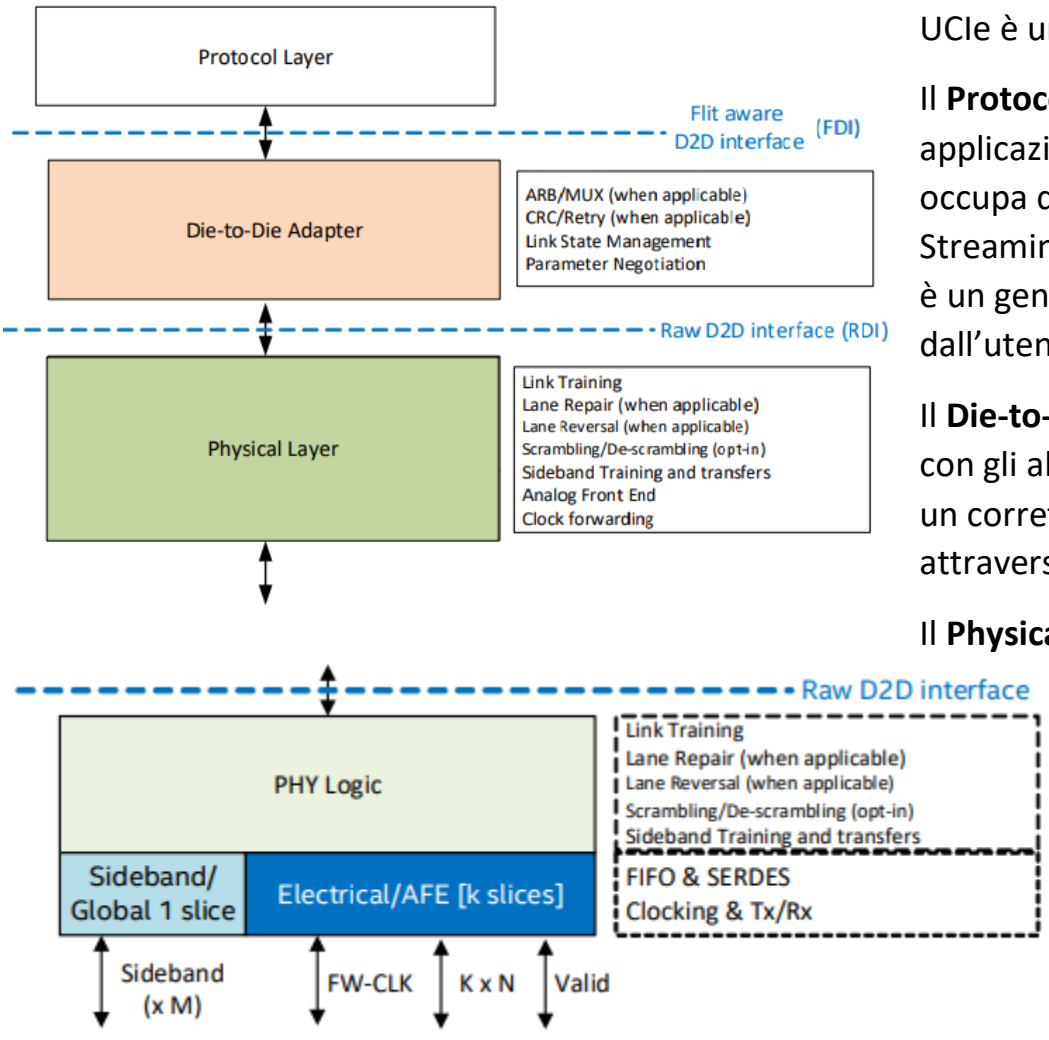
UCIe è un protocollo open che consente una interconnessione tra chiplet che sia: elevato bandwidth, bassa latenza, efficiente nei consumi e conveniente.

Consente di connettere chiplet di origine diversa, con diversi processi di fabbricazione, design diversi e diverse tecnologie di packaging.

Il passaggio da un solo chip monolitico a più chiplet interconnessi è dovuto a due motivi:

1. **Resa**, al crescere delle funzionalità di un chip aumenta la sua dimensione e complessità e conseguentemente anche la probabilità di difetti che rendono il chip inutilizzabile. Utilizzando chiplet si ha una resa percentuale più alta sullo stesso wafer. Prendere immagini da http://cloud.mooreelite.com/tools/die-yield-calculator/index.html
2. **Convenienza**, far produrre chip con un nodo avanzato (5 o 4 nanometri) è più costoso di uno meno recente (14 o 12 nanometri) e dato che alcuno unità non scalano bene (cache) o non necessitano di un PP avanzato, ha più senso usare PP più vecchi per queste unità.

Come



UCIe è un protocollo a livelli.

Il **Protocol Layer** varia da applicazione ad applicazione, si occupa di mappare CXL, PCIe e Streaming Protocol, quest'ultimo è un generico protocollo definito dall'utente.

Il **Die-to-Die Adapter** si coordina con gli altri due livelli, garantendo un corretto trasferimento dei dati attraverso il collegamento UCIe.

Il **Physical Layer** coordina le diverse funzioni e il loro corretto sequenziamento. Il Physical Layer organizza le linee in gruppi detti moduli, in base al numero di linee si parla di **Standard**

Package o **Advanced Package**. I moduli possono essere a loro volta raggruppati per avere maggiore parallelismo. La tabella riassume le principali differenze tra i due tipi di package.

	Standard Package	Advanced Package
Transfer rate GT/s	4, 8, 12, 16, 24, 32	
Numero di linee	16	64
Bump pitch (µm)	100-130	25-55
Lunghezza canale (mm)	≤ 25	≤ 2

La comunicazione tra i layer avviene tramite apposite interfacce definite da UCIe: FDI e RDI. Questo consente ai produttori di mischiare differenti strati da diversi fornitori, con un costo di integrazione più basso e in maniera più rapida.

Performance Target

	Transfer rate	Standard Package	Advanced Package
Bandwidth per lato del die (GB/s per mm)	4 GT/s	28	165
	8 GT/s	56	329
	12 GT/s	84	494
	16 GT/s	112	658
	24 GT/s	168	988
	32 GT/s	224	1317
Latenza (ns)		≤ 2	

Il transfer rate massimo, e conseguentemente il bandwidth, è direttamente proporzionale al bump pitch.

Protocol Layer

Questo strato si occupa di mappare i protocolli PCIe, CXL e qualsiasi Streaming Protocol definito dall'utente. I protocolli standard supportati sono: PCIe Gen5 e Gen6, CXL 2.0 e 3.0.

La caratteristica necessaria di un protocollo per essere mappato è il supporto al **FLIT**. Questa tecnologia ha lo scopo di garantire l'integrità del segnale e per fare ciò spezza lo stream in blocchi, su cui viene fatto il CRC, che poi verrà controllato dal destinatario per accertarsi dell'integrità dei dati. Può essere fatto con 68 o 256 Byte.

Questo strato prende i dati dai protocolli e li trasforma in pacchetti con un particolare formato che passa allo strato sottostante tramite l'interfaccia FDI.

//inserire screen

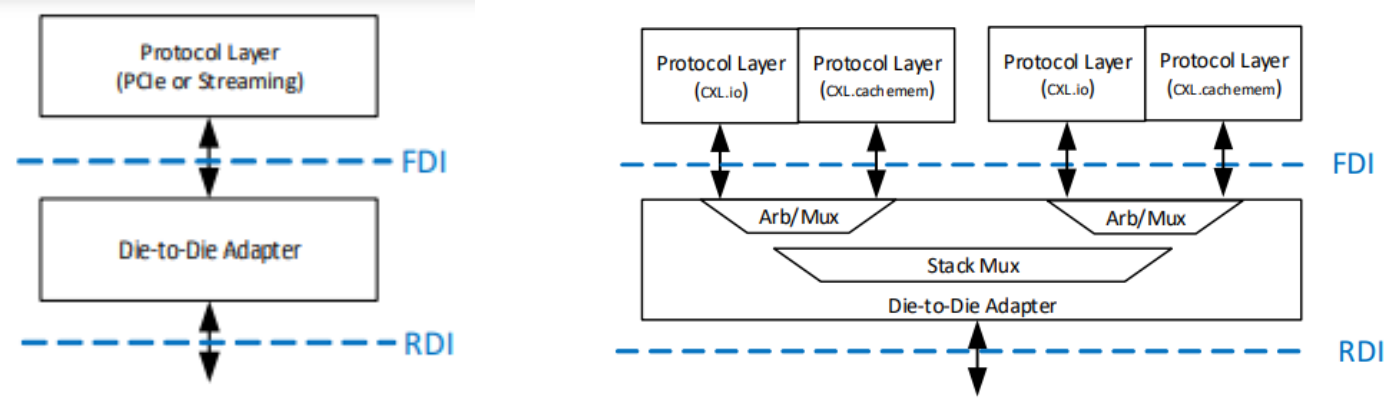
Die-to-Die Adapter

Questo strato si occupa di:

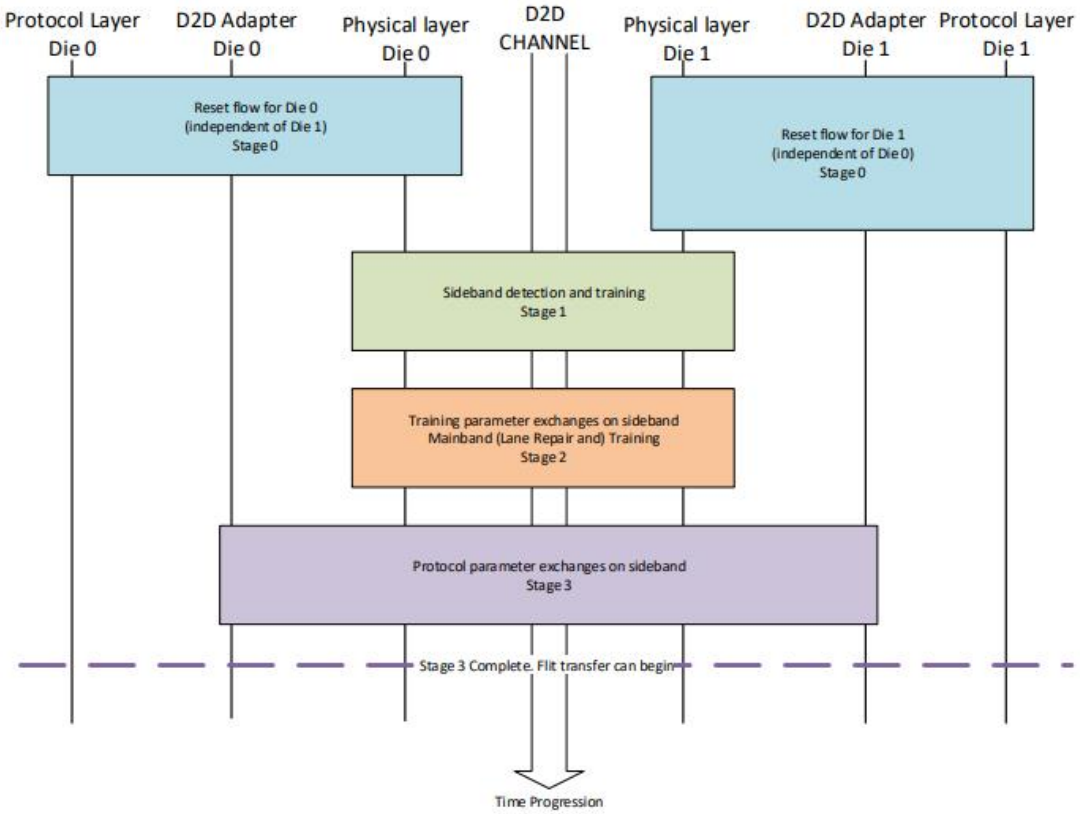
- Trasferire i dati in maniera affidabile;

- Gestire il MUX nel caso di più Protocol Layer;
- Gestire lo stato del collegamento;
- Negoziare il protocollo e i parametri del collegamento con il corrispettivo remoto.

Per quanto riguarda la topologia, ad un solo adattatore possono essere collegati uno o più livelli di protocollo tramite MUX.



Link Initialization



Prima di iniziare il trasferimento è necessario inizializzare il canale di comunicazione, questa fase è detta Link Initialization. Consiste di quattro fasi:

Stage 0: è specifico per ogni die e avviene in maniera indipendente, consiste nel fare il reset della configurazione precedente;

Stage 1: consiste nel cercare l'altro peer;

Stage 2: si scambiano i parametri di training ottenuti nello stage 1;

Stage 3: è la fase di negoziazione vera e propria, in cui le due parti decidono il protocollo da usare. È divisa in tre parti:

Parte 1: ogni adattatore determina i protocolli che supporta

Parte 2: le due parti si scambiano le capability e decidono il protocollo (PCIe, CXL) e i parametri (Flit Mode). Nel caso in cui si voglia usare un protocollo non standard, le due parti devono negoziarlo

Parte 3: si passa al Protocol Layer il risultato della parte 2 e può iniziare il trasferimento.

CRC

Il D2D Adapter si occupa di fare il CRC (Cycle Redundancy Check), che è un'operazione polinomiale fatta sui dati che produce una impronta, questa viene poi ricreata anche dal destinatario e le confronta. Questa operazione ha lo scopo di individuare possibili errori dovuti al rumore nel canale di trasmissione. In particolare, il blocco elaborato deve avere una lunghezza di 128B, se non viene raggiunta tale lunghezza si mette un padding di '0'.

Retry

Nel caso in cui il CRC generato dal destinatario non coincida con quello ricevuto, il D2D Adapter sfrutta il meccanismo di retry definito dallo standard PCIe, al fine di ricevere un pacchetto corretto.

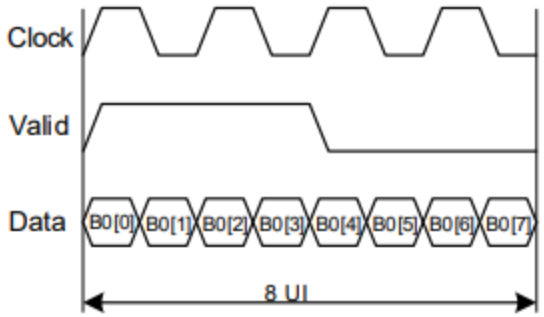
Logical Physical Layer

Questo livello si occupa di:

- Inizializzazione e training del collegamento
- Power management
- Fare il mapping byte-linea per trasmettere i dati sulle linee
- Trasmettere e ricevere i messaggi sulla sideband

Mapping byte-linea

I pacchetti sono trasmessi in byte, di ogni byte il bit 0 è trasmesso per primo, sotto è riportato un esempio:



Ogni byte è trasmesso su una linea diversa: il byte 0 sulla linea 0, il byte 1 sulla linea 1 e così via.

Il segnale “valid” è usato per distinguere un byte da un altro, è asserito per i primi 4 bit e poi de-asserito per gli ultimi 4. Questo consente il trasferimento utilizzando le modalità messe a disposizione dal Protocol Layer.
La trasmissione su sideband differisce, infatti è possibile trasmettere pacchetti di 32 o 64 bit di dati.

Data Lane repair

Nel caso una linea sia danneggiata è possibile fare il remapping per spostare il traffico su linee non utilizzate, questo è possibile anche per il segnale di clock. Nel caso di Standard Package tutte le linee sono utilizzate e quindi l’unica soluzione è quella di mappare il modulo come x8.

Sideband

Il collegamento tramite sideband si utilizza per il training, gestione e scambio di parametri del canale o per accedere ai registri sul die remoto. L’accesso ai registri avviene in maniera indiretta a livello del Protocollo e queste richieste passano dal sideband.

Sono possibili tre tipi di messaggi:

- Register access, si possono usare per la configurazione o RW sulla base dell’indirizzo.
- Messaggi senza dati, si usano per la gestione del collegamento
- Messaggi con dati, vengono usati per lo scambio di parametri o link training

High level Software view of UCIe

UCIe è consistente con il protocollo che mappa, quindi una porta UCIe che mappa CXL viene vista come una normale porta CXL e lo stesso vale per PCIe. Questo viene fatto per mantenere la retrocompatibilità con gli standard pre-UCIe.

Software discovery of UCIe links

Sia software che firmware UCIe-aware sono in grado di scoprire la presenza di connessioni UCIe. L’host prevede un blocco di registri chiamato CiRB che raccoglie le capacità del collegamento UCIe.

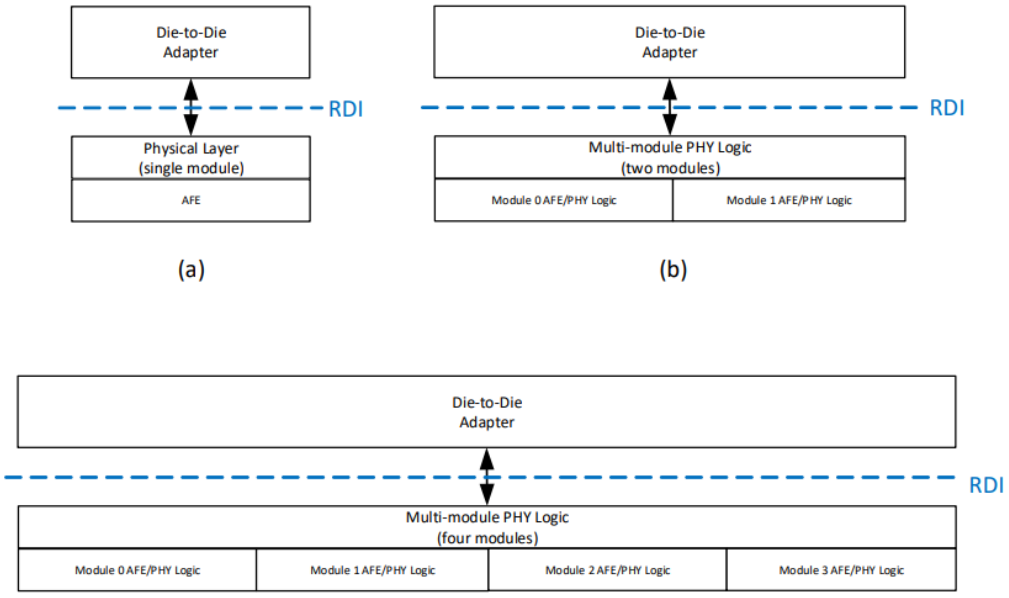
Per definire le capacità UCIe di un dispositivo si usa DVSEC che è una struttura dati interoperabile tra più vendor la usano per definire quali sono i registri e un’interfaccia di programmazione che altri vendor possono mettere sul loro silicio.

Definizione delle interfacce

Per la comunicazione tra i livelli proposti dal protocollo esistono delle apposite interfacce:

- **RDI** (Raw D2D Interface) che sta PHY e D2D
- **FDI** (Flit-aware D2D Interface) che sta D2D e Protocol Layer

RDI

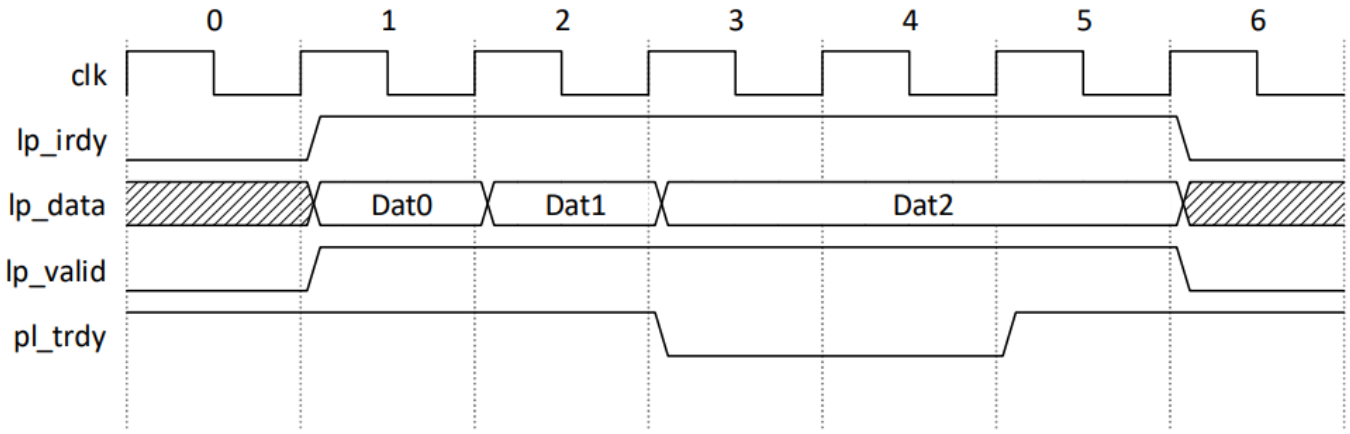


È un collegamento 1 a 1 tra D2D e la logica del PHY, questa logica si occupa di nascondere i moduli sottostanti.

La comunicazione avviene in maniera bidirezionale tra D2D e PHY. Quelli da PHY a D2D sono nella forma pl_*, mentre quelli da D2D a PHY sono del tipo lp_*.

La condizione necessaria per la comunicazione è che entrambe la parti abbiano lo stesso clock ed i segnali inviati siano sincroni con il clock (segnale lclk).

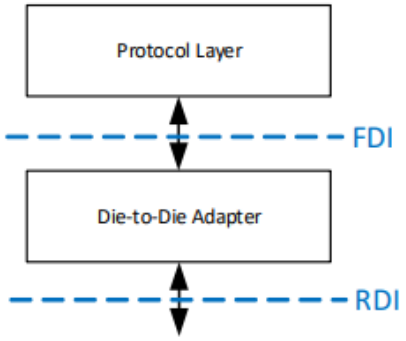
Il **trasferimento dati da D2D a PHY** avviene come mostrato in figura:



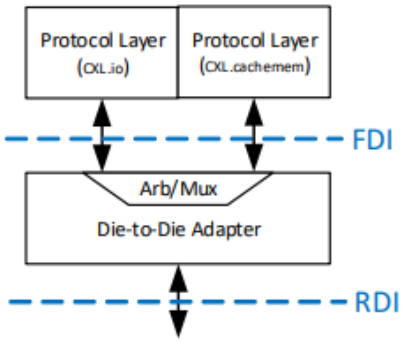
In cui lp_irdy indica che D2D deve inviare i dati, pl_trdy indica che PHY è pronto a ricevere i dati, lp_valid indica che i dati sono validi. I dati viaggiano sul bus lp_data.

FDI

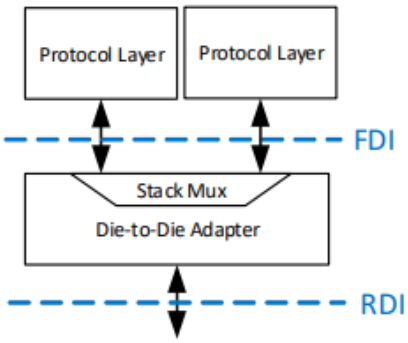
Questa interfaccia presenta diverse possibili topologie:

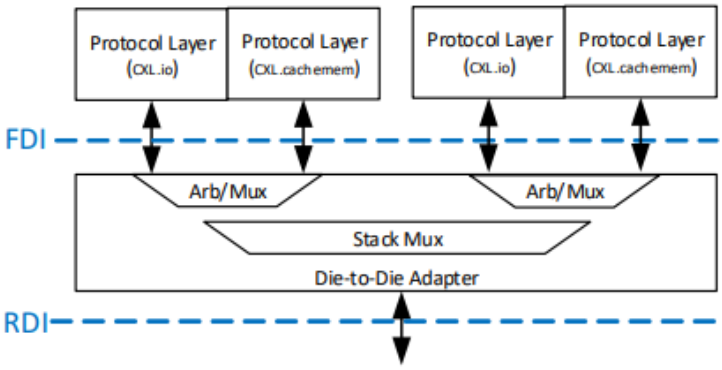


(a) Single Protocol



(b) Single CXL stack

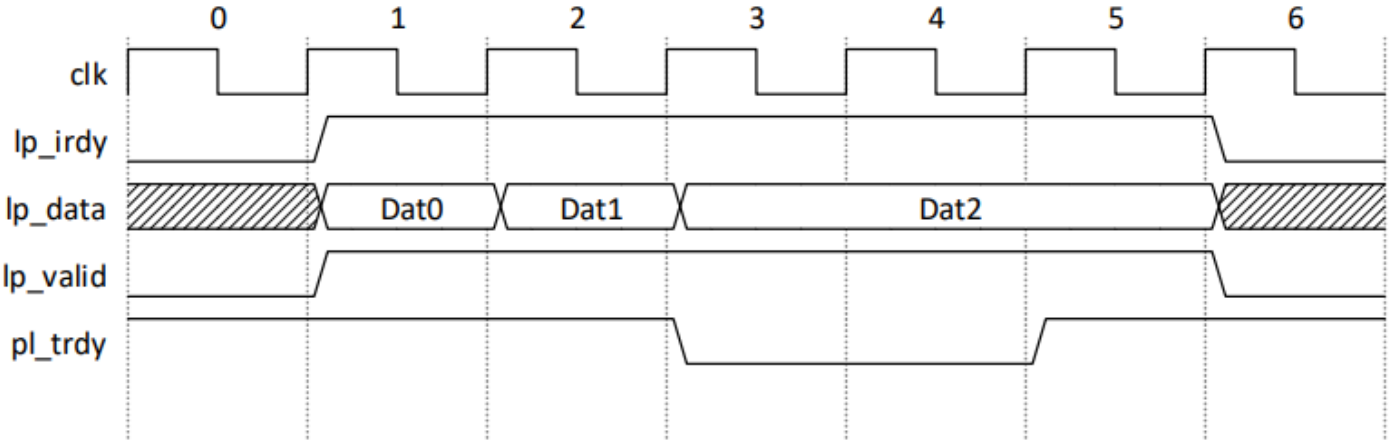




La nomenclatura dei segnali è analoga a quella di RDI, solo che ne vengono aggiunti altri relativi al FLIT, come ad esempio quelli per segnalare un CRC non valido.

Entrambi i lati dell'interfaccia devono operare allo stesso clock.

Il **trasferimento di dati da Protocol Layer a D2D Adapter** avviene come segue:



Dato che questo livello è Flit-aware, è necessaria la gestione dei DLLP (Data Link Layer Packet), che servono per ack, flow control e power management. La loro ricezione e invio sono gestiti con gli appositi segnali predisposti.

CCIX

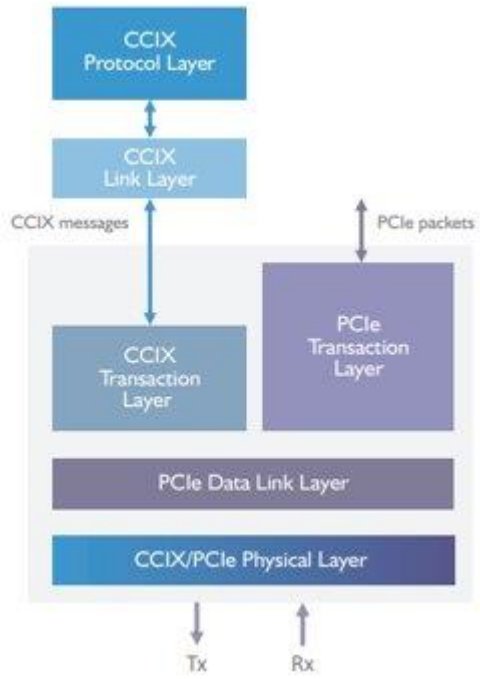
CCIX è un protocollo di interconnessione che consente di trasmettere i dati tra più chip. L'infrastruttura sottostante è quella di PCIe, a cui aggiunge la coerenza delle cache. Il protocollo prevede che i chip siano tra loro pari, inoltre consente più topologie, ad esempio anche il daisy chaining, ma anche a switch come su CXL.

L'obiettivo di CCIX è ridurre la latenza che per PCIe non è indifferente.

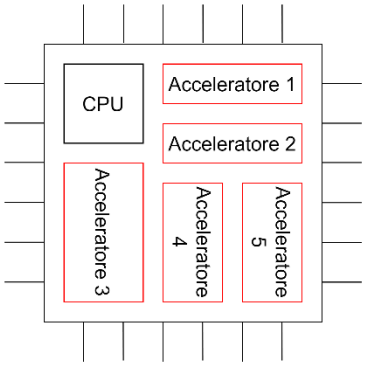
Architettura

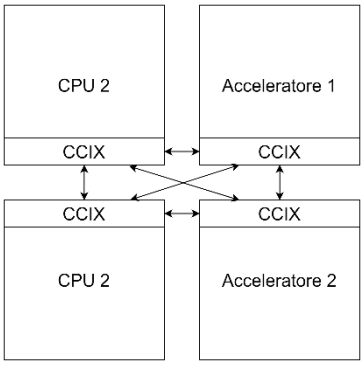
L'architettura di CCIX è su più livelli:

- **Protocol Layer**, si occupa di fare le operazioni di lettura e scrittura, garantisce la coerenza delle cache.
- **Link Layer**, crea i pacchetti e fornisce la possibilità di concatenare più messaggi per avere una maggiore efficienza.
- **Transaction Layer**, gestisce il flusso di byte tra il Link Layer e quello fisico, si occupa della gestione dei crediti necessari per il controllo di flusso.
- **Data Link Layer**, si occupa di gestire l'error correction e il retry.
- **Physical Layer**, si occupa di gestire la connessione fisica, che può essere chip-to-chip o die-to-die (da CCIX 2.0).



Per una connessione die-to-die il Physical Layer può funzionare sia in maniera sequenziale che parallela. Nel caso di connessione tra chip (a livello di sistema), sfrutta il livello fisico di PCIe.





Coherency Protocol Layer

Per la gestione della coerenza sono necessari più **agenti**:

- **Request Agent**, è un core della CPU che si occupa di mandare le richieste.
- **Home Agent**, si occupa di gestire la coerenza tra diverse linee di cache.
- **Slave Agent**, è il memory controller che sta oltre il collegamento CCIX.
- **Error Agent**, riceve e logga i messaggi di errore.

Request Agent

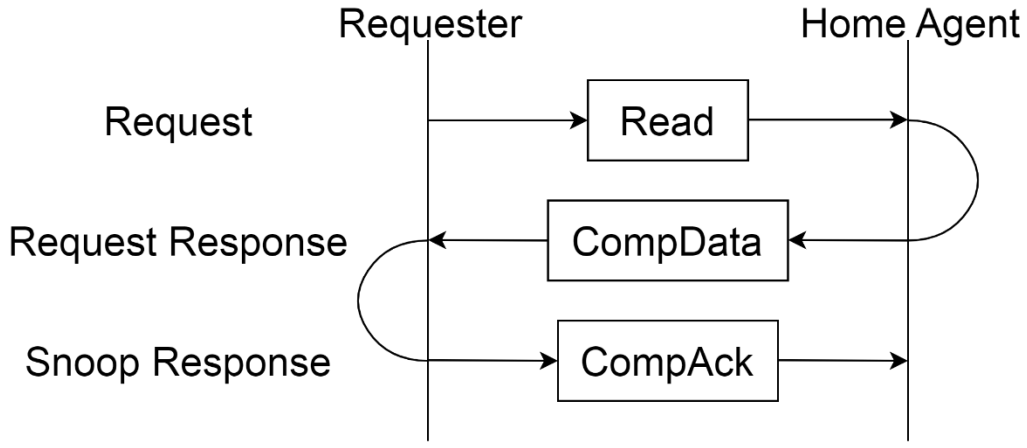
È l'entità che si occupa di fare le richieste, che possono essere:

- **Read**, ottiene la copia della linea.
- **Write**, passa la copia dirty della linea.
- **Atomic**, spostano l'operazione in memoria.
- **Dataless**, modifica lo stato di una linea di cache senza inviare dati.

Operazioni di Read

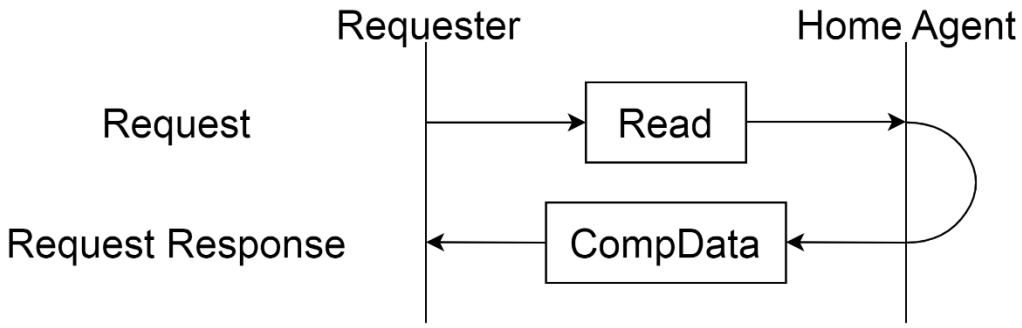
Si dividono a loro volta in due categorie:

- **Coerenti**, generano snoop e la linea viene cercata in altre cache. Di questo tipo ci sono:
  - **ReadUnique**, ottiene una copia unica della cache, invalidando le linee nelle altre cache.
  - **ReadClean**, **ReadNotSharedDirty**, **ReadShared**, queste operazioni modificano lo stato della linea.
  - **ReadOnce**, **ReadOnceCleanInvalid**, **ReadOnceMakeInvalid**, scattano un'istantanea di un'area di memoria coerente, vengono usate dai dispositivi di I/O, prevedono delle operazioni per verificare lo stato della linea dopo la transazione.



- **Non Coerenti**, non generano snoop. C'è:

- **ReadNoSnp**, si usa per l'accesso alle periferiche, perché non ha senso cercare leggere in cache, almeno al primo accesso.



Operazioni di Write

Anche in questo caso ci sono operazioni coerenti e non coerenti, analogamente alle read.

Tra le **operazioni coerenti** ci sono:

- **WriteBack**, scrive una linea dirty in RAM e la de-alloca dalla cache, si usa in caso di eviction.
- **WriteClean**, scrive la linea dirty in RAM e diventa clean.
- **WriteEvict**, sposta la linea in una cache più lenta (L2 o L3), utilizzata per migliorare le performance.
- **WriteUnique**, si usa quando l'I/O non conserva copie in cache, genera degli snoop per verificare che la linea sia la più recente.

Tra le **operazioni non coerenti** c'è:

- **WriteNoSnp**, si usa per scrivere sulle periferiche.

Operazioni Atomiche

Questo tipo di transazioni spostano l'operazione verso il dato. Ad esempio: se voglio fare un'operazione tra due valori, anziché prenderli dalla memoria, caricarli nei registri e fare l'operazione, si indicano gli indirizzi su cui fare l'operazione, che poi verrà eseguita e il risultato messo o in RAM o in un registro.

Il secondo utilizzo di queste operazioni è il sequenziamento delle operazioni svolte dagli agent verso la memoria. Esistono 4 tipi di operazioni atomiche:

- **Atomic Store**, si inviano i dati e si riceve una semplice risposta.
- **Atomic Load**, si inviano i dati per l'operazione e si riceve il dato prima dell'operazione, questo perché nel caso in cui l'operazione sia unidirezionale non è possibile risalire al dato originale.
- **Atomic Swap**, cambia il nuovo dato con uno vecchio.

- **Atomic Compare**, sono inviati due dati, uno viene confrontato con quello in memoria e se combaciano il dato viene salvato in memoria.

Operazioni Dataless

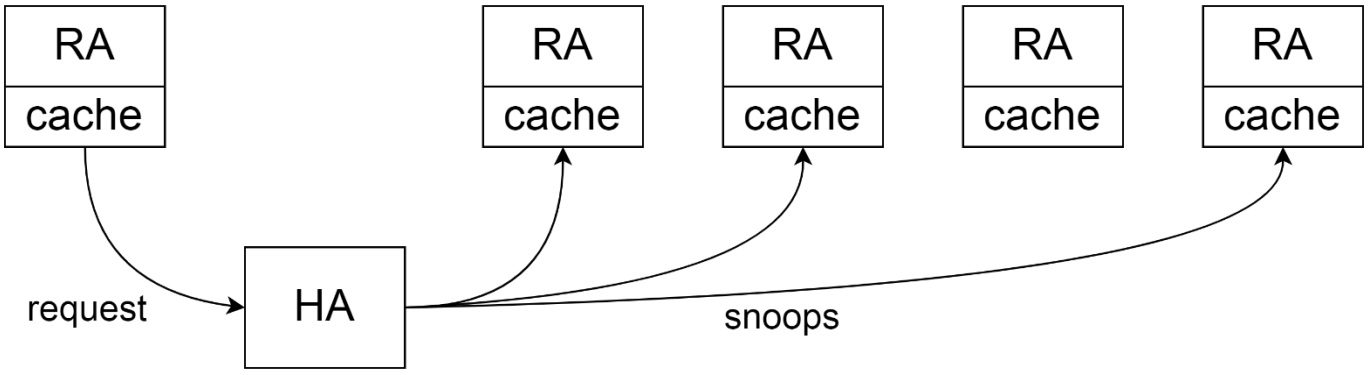
Queste transazioni consentono di cambiare lo stato di una linea di cache senza spostare dati. Esistono diverse operazioni:

- **CleanUnique**, **MakeUnique**, si usa quando il Request Agent vuole cambiare lo stato di una linea di cache da Shared a Unique.
- **CleanShared**, **CleanInvalid**, **MakeInvalid**, si usa per la gestione della cache, utilizzati quando il sistema usa sia HW che SW coherency.
- **CleanSharedPersist**, si usa per assicurarsi che una linea dirty sia salvata in memoria non volatile
- **Evict**, notifica l'Home Agent che il Request Agent non ha più una certa linea.

Home Agent

L'Home Agent si occupa di controllare tutte le cache del sistema quando una linea viene acceduta. Per sapere quale dei Request Agent ha una certa linea, l'Home Agent può utilizzare uno **Snoop Filter** o una **Directory**, in alternativa è possibile mandare le richieste in broadcast, ma risulta meno efficiente.

L'Home Agent si occupa di inviare gli snoop ai Request Agent, nel caso in cui la linea richiesta non sia in nessuna cache all'interno del sistema, l'Home Agent accede in RAM per leggere la linea richiesta.



Tipi di Snoop

Gli snoop hanno lo scopo di modificare lo stato delle linee di cache. Si indica con Snoopee un Request Agent che riceve uno snoop.

La specifica prevede due comportamenti:

- **Expected**, è il miglior cambio di stato.
- **Permitted**, è il cambio di stato permesso, ma non quello ottimale.

Sta allo snoopee decidere quale comportamento seguire.

I tipi di snoop sono:

- **SnpToI**, invalida la linea che viene rimossa dalla cache, si utilizza quando un agente vuole modificare la linea.
- **SnpToS**, la linea va in shared, si usa quando un altro agent legge la linea.
- **SnpToSC, la linea va in shared clean, previene l'uso di shared dirty.**
- **SnpToC**, la linea va in uno stato clean, si usa per estrarre una linea dirty.
- **SnpToAny**, la linea rimane nello stesso stato, si usa per ottenere un'istantanea della linea.
- **SnpMakeI**, forza l'invalidazione della linea, scarta i dati dirty.

Slave Agent

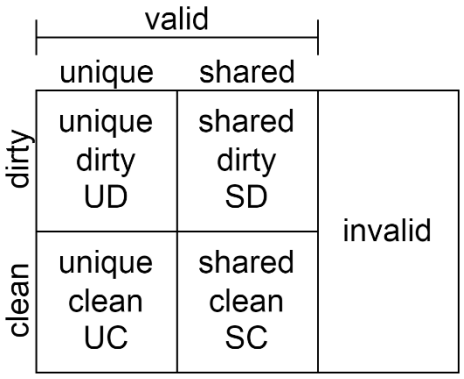
Si usa per aumentare la memoria disponibile per l'host, solitamente si trova su un altro chip. L'Home Agent accede al memory controller dello Slave Agent.

Protocol Overview

Una CPU può prendere dati da salvare in cache da una memoria non direttamente collegata alla CPU stessa, per fare ciò si deve fare una richiesta all'Home Agent (HA) della CPU remota, a cui la memoria è direttamente collegata. La richiesta verso l'HA viene fatta dal Request Agent.

Il protocollo di coerenza utilizzato è non-MESI e prevede cinque stati:

- **Unique Dirty** (UD), la linea è in una sola cache ed è modificata (analogo a Modified).
- **Unique Clean** (UC), la linea è in una sola cache clean (analogo a Exclusive).
- **Shared Clean** (SC), la linea è in più cache clean (analogo a Shared).
- **Shared Dirty** (SD), la linea è in più cache, in una è dirty, mentre nelle altre è clean. Questo stato serve al momento dell'**eviction**, infatti solo la cache che è in SD dovrà mandare la linea all'Home Agent per sovrascriverla in memoria.
- **Invalid**, la linea non è in cache.



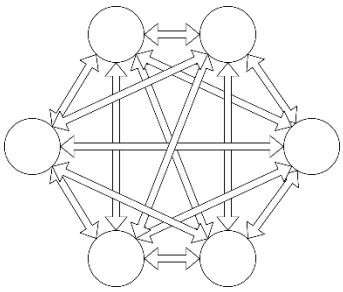
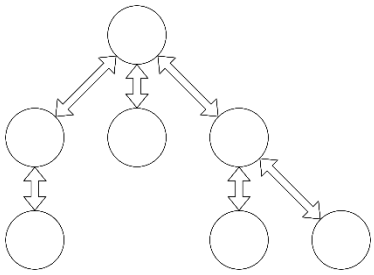
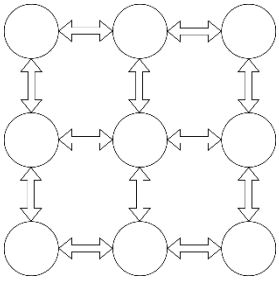
Inoltre, sono supportate **linee di cache parziali**, per le quali ci sono ulteriori stati:

- **Unique Clean Empty** (UCE), indica una linea di cache senza dati.
- **Unique Dirty Partial** (UDP), indica che sono stati modificati alcuni dati dalla linea, ma non tutti.

Topologie

CCIX non prevede una topologia precisa, ne sono proposte alcune poiché sono conosciuti algoritmi di routing che non vanno in deadlock. Tra queste ci sono:

- **Fully connected**, ogni chip è connesso agli altri.
- **Tree**, ha struttura gerarchica come quella di PCIe.
- **Multi-dimensional Array**, non ci sono restrizioni per le dimensioni e il numero di agenti.

Differenze tra CXL e CCIX

	CCIX	**CXL**
Modello	Simmetrico, chip come peer	Asimmetrico, CPU master e acceleratore slave
Dispositivi connessi	CPU e CPU, CPU e acceleratore, acceleratore e acceleratore	CPU e acceleratore
Network	Ogni chip ha una cache coerente e una directory, che devono essere sincronizzate tra loro per far sì che la coerenza sia mantenuta	L'acceleratore ha un'architettura fortemente parallela e ha una architettura diversa dalla CPU
Complessità HW	Richiede una circuiteria più complessa su tutti i chip	Richiede una circuiteria meno complessa, perché alcuna unità sono solo su master
Use case	AI, automotive, edge processing	Elaborazione big data
Consumo	Pensato per ambienti automotive	Pensato per ambienti data center
Memorie utilizzate	LPDDR, latenza bassa e bandwidth ridotto	HBM, alto bandwidth, latenza in secondo piano