**CXL**

**Introduzione**

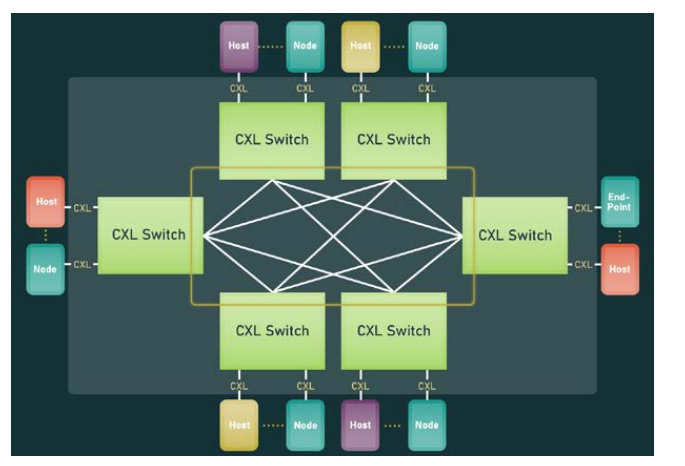
CXL è un protocollo che nasce per consentire la comunicazione tra CPU e altre periferiche, sfrutta le stesse connessioni fisiche di PCIe, fornendo in più la coerenza.

CXL opera a livello hardware, riducendo l’overhead dove all’intervento di un software.

È costituito da tre protocolli: CXL.io, CXL.cache e CXL.mem.

CXL prevede il memory pooling and sharing: questi concetti consentono a più host di accedere ad una stessa area di memoria con i dati più aggiornati senza interventi software.

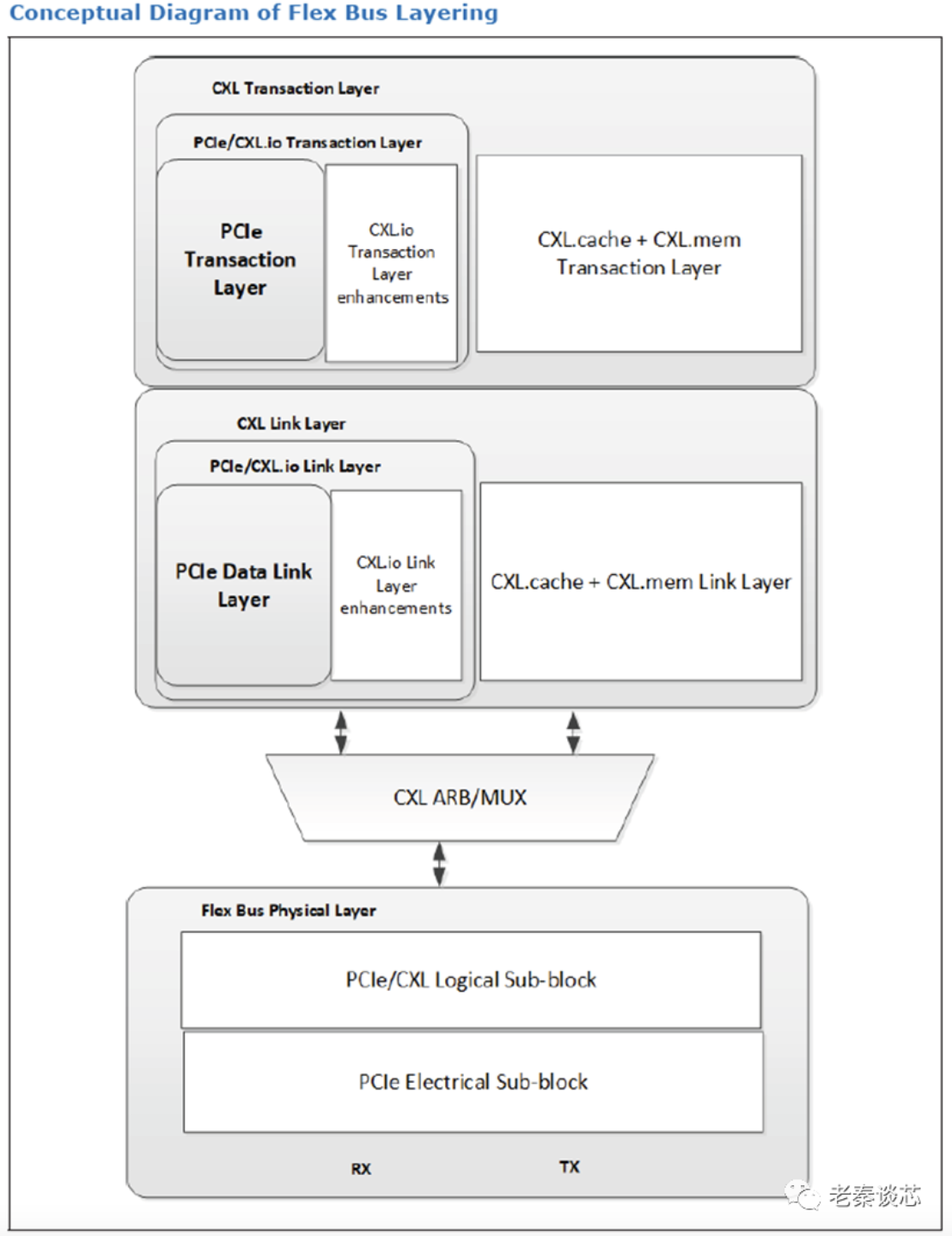
CXL 3.0 introduce una nuova topologia, detta *fabric* che non è più gerarchica e ad albero come per PCIe:



Il CXL *fabric* supporta fino a 4096 nodi che comunicano con un meccanismo di indirizzamento chiamato *Port Based Routing* (PBR).

Con il termine nodo si intende una CPU, un dispositivo PCIe o un acceleratore.

CXL prevede il *Flex Bus* (FB), che è una porta che permette di scegliere tra il protocollo nativo PCIe o CXL. Questa scelta viene fatta durante l’inizializzazione del collegamento. Il FB è diviso in due layer, uno che gestisce la logica di transaction e uno il link.



I protocolli CLX.cache e CXL.mem sono aggregati insieme nei due livelli.

**Cache coherency in CXL**

La coerenza è garantita dai tre protocolli di CXL:

* CXL.io, consente all’host di vedere la memoria dell’acceleratore come se fosse la sua;
* CXL.cache, consente di avere coerenza e comunicazioni veloci tra host e CXL device;
* CXL.mem, consente l’accesso alla memoria dell’acceleratore senza l’uso di SW dedicato.

**CXL.io**

È basato su PCIe e viene usato dall’host per: cercare, enumerare, configurare e gestire un qualsiasi dispositivo CXL.

Fornisce una interfaccia non coerente per fare load e store per i dispositivi di I/O. Le varie regole si basano sulla definizione del transaction layer definito da PCIe.

**CXL.mem**

Consente all’HA dell’host di accedere alla memoria dell’acceleratore senza l’uso di SW aggiuntivo. Ad alto livello la memoria dell’acceleratore viene vista come la normale memoria DDR dell’host, la latenza è minima. Le operazioni svolte sono molto semplici e veloci, sono lettura e scrittura.

È l’interfaccia transazionale tra CPU e memoria, usa il livello fisico e il link layer per comunicare tra i die.

Permette tre modelli di coerenza:

* HDM-H
* HDM-D
* HDM-DB

Sfrutta il modello Master-Subordinate, in cui il master non è altro che il coherency engine e il dispositivo il subordinate.

Le transazioni posso essere da M a S o viceversa.

Le transazioni M2S possono essere di tre classi:

* Request w/out data;
* Request w/ data;
* Back-invalidation response.

Le transazioni S2M possono essere:

* Response w/out data;
* Response w/ data;
* Back-invalidation snoop.

I canali per la comunicazione sono 6, unidirezionali e indipendenti.

Il back-Invalidation snoop è un canale che consente la modifica dello stato della cache dell’host.

**CXL.cache**

Serve ad avere coerenza e comunicazioni veloci tra host e dispositivi CXL. L’HA sfrutta il bus proprietario dell’host per interagire con la cache di più basso livello, che usa CXL per interagire con gli altri livelli di cache core della CPU e dispositivi CXL. In tal modo si realizza una gerarchia ed ogni livello vede l’attività del proprio dominio. Questo protocollo consente l’accesso da parte del CXL device alla memoria dell’host, così host e dispositivo lavorano su dati condivisi in maniera coordinata.

L’HA si occupa di controllare le operazioni e impedire cambi simultanei, inoltre quando viene fatta una modifica questa viene propagata a tutte quante le copie.

Definisce le interazioni tra host e dispositivi, l’interazione è di tipo request-response: ad una richiesta riceve una risposta e in alcuni casi c’è un trasferimento di dati.

La comunicazione avviene su canali unidirezionali: D2H (device-to-host) e H2D (host-to-device). I canali sono indipendenti e permettono un solo tipo di messaggio.

Le richieste di tipo D2H riguardano la memoria e possono ricevere 0, 1, 2 risposte e al più una linea di cache di 64B.

Le richieste H2D sono snoop per mantenere la coerenza.

Come detto precedentemente, i canali sono indipendenti, ma c’è bisogno di ordinamento che viene fatto con i messaggi di Global Observation, inviati dall’host al dispositivo.

Il protocollo prevede quattro possibili semantiche:

* CXL.cache Read: riceve una linea di cache
* CXL.cache Read0: non riceve una linea di cache
* CXL.cache Write: invia dei dati, ricevendo un ack dall’host
* CXL.cache Read0-Write: invia dei dati, ma senza ricevere ack dall’host

Le richieste sono di vario tipo e ognuna ha una specifica funzionalità, sono riportati alcuni esempi:

* RdCurr, lettura di una linea di cache con i dati più aggiornati;
* RdOwn, lettura di una linea di cache da salvare in stato E o M;
* RdShared, lettura di una linea di cache da salvare in stato S;
* CleanEvict, si chiede di fare eviction di una linea in stato E.

**Aspetti Software**

Lo scopo di CXL è rendere più facile la programmazione, creando e gestendo un’astrazione che fa vedere al programmatore un sistema a memoria comune, anziché a scambio di messaggi.

**CXL devices**

I CXL device che 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:

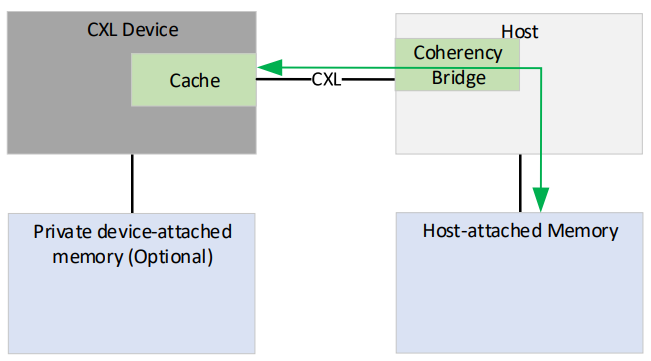
* Host-only coherent (HDM-H)
* Device coherent (HDM-D)
* Device coherent using Back-Invalidation Snoop (HDM-DB)

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.

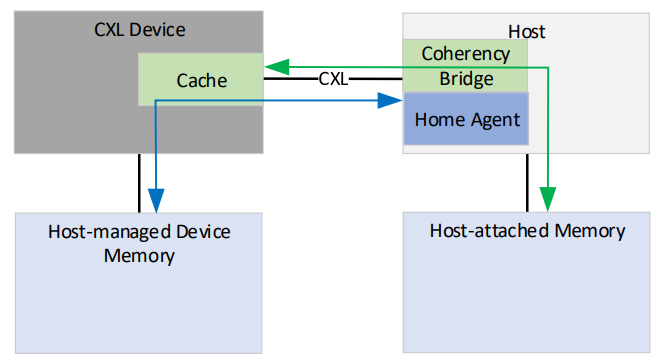
**CXL type 1 device**

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 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.