PCI expressを進化させた次世代インターコネクト規格CXLの紹介

2022/4/1

富士通株式会社
Linuxソフトウェア事業部
五島康文

FUJITSU

© 2022 Fujitsu Limited

初めに

FUJITSU

- CXLが新たなインターコネクトの仕様として注目されつつある
  - GPGPU, Smart NIC, FPGA, メモリ(揮発性、不揮発性）など、次世代のデバイスを高速に接続
- 一方で、CXLの仕様は過去の仕様や技術資産の蓄積の上に成り立っているため、若手技術者には非常にとっつきにくいものとなっている
  - 過去のことを知らないと、仕様書中の言葉すらもわからない可能性がある
  - CXLの仕様書はPCI, ACPIの用語が多々出てきており、そちらをポイントしているケースも多い
    - 上記仕様について経験のない若手技術者にとって、非常にハードルが高い仕様となっている
- このため、本資料では読者が以下になることを目指す
  - CXLの機能概略を知ることができる
  - CXLの仕様書が自力で読める(*)ようになる
    - CXL対応デバイスやそのドライバの開発のため
    - 対応OS・プラットフォームの開発やトラブル時のサポートのため

    (*) 「スラスラと読んで理解できる」までいかなくとも、「仕様書を“眺めて”みようか」と一人でも思ってもらえればOK!
    - 上記のために、前提となるPCI, ACPIなどについても概略を説明する

© 2022 Fujitsu Limited

おことわり

FUJITSU

- 誤りがないよう努力していますが、筆者が誤解している個所や不正確な個所がないとは言い切れません
  - CXLおよびその関連仕様書はかなりのページ数であり、筆者もまだまだごく一部の箇所しか読めていません
    - CXL ver2.0仕様本体だけで628page
    - PCIe Gen5本体で1299page
    - ACPI ver.6.4で1063page
    - その他まだCXL仕様本体にマージされていない仕様や実装のガイドラインなども色々
  - 仕様の原文を確認するようにしてください
    - 誤りがあれば、ご指摘いただけると幸いです
- 説明は特に断りがなければv2.0の仕様ベースになります

© 2022 Fujitsu Limited

内容

FUJITSU

- CXLの概略：
  (概略だけわかればよいという人はここを読むだけでOK)
- CXLの仕様書を理解するための前提知識 ~より深く知るために~
  - PCI/PCI express
  - ACPI
- CXLの仕様
  - レイヤー構造
  - configuration空間
  - メッセージ
  - ACPIの新仕様
- CXLにおける(不揮発)メモリの扱い
- まとめ

© 2022 Fujitsu Limited

CXLの概略

FUJITSU

© 2022 Fujitsu Limited

CXLとは

FUJITSU

○ 新たなインターコネクトの規格

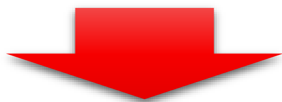
- Compute eXpress Linkの略で、"Open industry standard for high bandwidth, low-latency interconnect" とされている
- 現在主流のインターコネクトであるPCI express(以後PCIe) の仕様を流用 (PCIe Gen.5以降が条件)
- 特にGPGPU, Smart NIC, FPGA, メモリ(揮発性、不揮発性)のようなデバイスの接続でメリット
- 現在の仕様はver 2.0
- CXLのBoard of Directorの企業 (2022年1月調べ)
  - Alibaba, AMD, ARM, CISCO, DELL, Google, HPE, Huawei, intel, Meta(facebook), MicroSoft, nvidia, SAMSUNG, XILINX
- CXLと目的が近いライバルの仕様としては、CCIX、OpenCAPIが存在
  - しかし、 (ContributorやAdopterの数を含めても) 参加企業数はCXLと比べると少ない
    - 青地は二股している企業、赤字は全てに最上級メンバとして参加している企業
    - CCIX:
      - AMD, ARM, HUAWEI, Mellanox(現nvidia), Qualcomm, XILINXがPromoterとして参加
    - OpenCAPI:
      - IBM, XILINXがStrategic Levelとして参加
- 2021年にはGen-Zコンソーシアムも吸収し、システム内部だけでなくシステム外部への接続も視野に

© 2022 Fujitsu Limited

CXLが生まれた背景

FUJITSU

- データの高速な処理の必要性の増大
  - 機械学習などの現在の技術トレンドの影響
- CPUの処理速度の頭打ちによる処理のオフロードの必要性
  - GPGPU、FPGA、Smart NICなどによるデータ処理の肩代わり
- メモリの容量の増大の必要性
  - DRAMは素子構造上容量限界が見えてきている

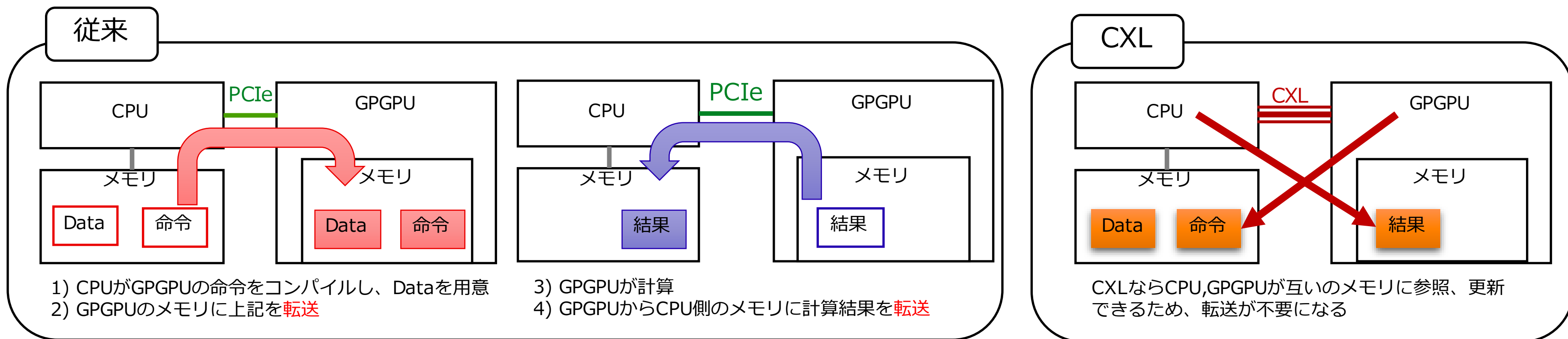


PCI expressに代わる、デバイスやメモリを接続する
新たなインターコネクトが必要

© 2022 Fujitsu Limited

CXLによって何が嬉しいのか？

- 例) GPGPUなどの計算が効率的に行える
  - 現状では、CPUが使うメモリ(DRAM)とGPGPUの”デバイス内の”メモリ間で、互いにデータ転送が必要(*)
    - 計算対象のデータだけでなく、GPGPUに実行させる命令の転送も必要で、面倒なうえ時間がかかる
  - CXLではCPU,GPGPUが、お互いのメモリをinteractiveに直接参照・更新できる
    - 機械学習などの計算が効率よく行えると考えられる
  - FPGAやSmartNICによるデータ処理のオフロードについても同様



- また、DDRxで接続するDRAMだけでは足りなってきたメモリ容量の増設なども視野

(*) 参考：CUDA Programming https://www.sintef.no/globalassets/upload/ikt/9011/simoslo/evita/2008/seland.pdf

© 2022 Fujitsu Limited

互いのメモリへのアクセスを効率的に

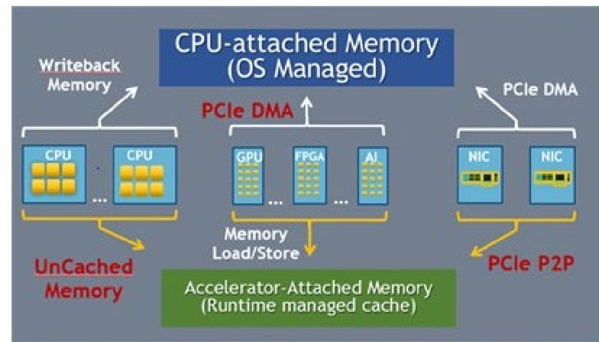
FUJITSU

- 互いのメモリをインタラクティブに更新するためにはCacheを使ったアクセスが不可欠
  - PCIeではCPUやデバイスのCacheを使ったアクセスはできない
    - デバイスのメモリ空間をホストの空間にマップしてもWrite Through のアクセスしかできず、遅い
    - DMAのような一括したデータ転送を行うことが前提であり、互いにinteractiveなメモリ参照、更新ができない
  - 互いにCacheを使ったinteractiveなアクセスをしたい
    - 必要に応じてWritebackさせたい

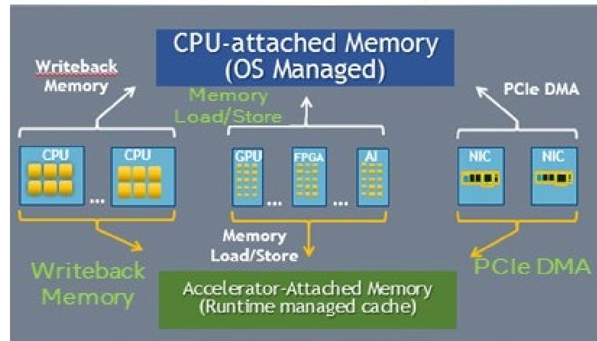
CXLは上記を実現する仕様と言える

  - 逆に、interactionが少ないデバイスでは、CXLの恩恵は受けにくい
    - データを一括処理してDMA転送し、終わったらCPUに割り込みをかける…とかの程度の機能のデバイスならあまり益はないとされる

右図は以下より抜粋(出典元はIntelのkeynoteの模様)
https://www.servethehome.com/wp-content/uploads/2021/08/Intel-Hot-Interconnects-2021-CXL-1-Open-Interconnect.jpg



With PCIe-only



CXL Enabled Environment

© 2022 Fujitsu Limited

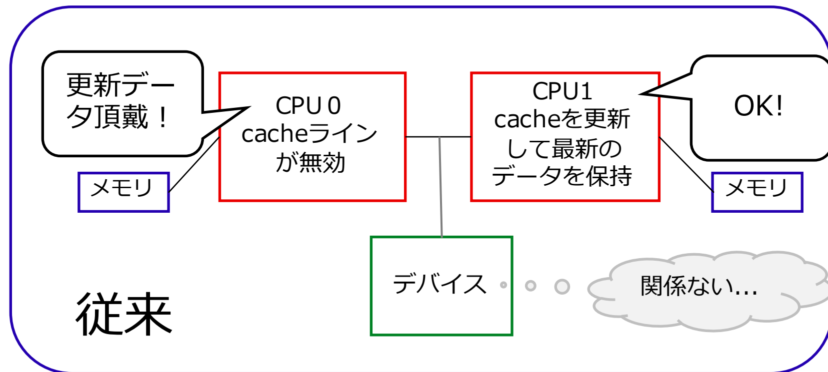
Cache-Coherentなinterconnect

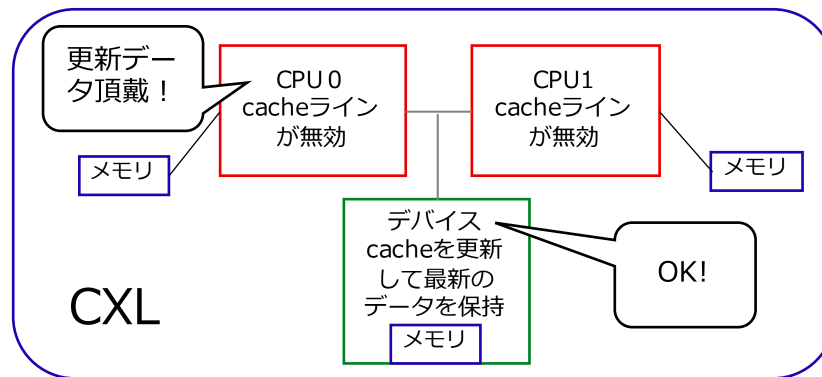
FUJITSU

- CXLのWebページ(*)ではCXLを一言で以下のように説明
  - “Compute Express Link™ (CXL™) is an industry-supported Cache-Coherent Interconnect for Processors, Memory Expansion and Accelerators. “

    ⇒ ここでいう「Cache Coherent とは？」
- これまではCPUの間だけで、メインメモリに対するキャッシュの内容について、互いに調停すればよかった
  - MESI, MOESIなどのcache coherenceプロトコルが利用されている
- 前述のCacheでの互いのアクセスを実現するためには、デバイスとCPUの間でも、同様の調停を行う必要
- CXLではMESIプロトコルを採用してこれを実現

状態	意味
Modified	メモリが変更され、当該キャッシュにのみ最新データが存在
Exclusive	当該キャッシュにのみ最新データが存在しているが、メモリとはデータが一致(clean)
Shared	ほかのキャッシュにもデータが存在し、メモリとデータは一致
Invalid	このキャッシュラインは無効



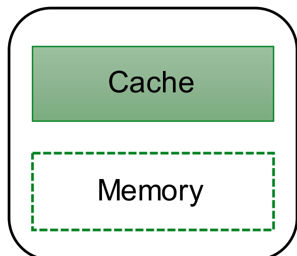


© 2022 Fujitsu Limited

CXLの特徴

- Gen 5.0以降のPCIeを流用
  - 接続(物理層)はPCIeそのままにしつつ、その上位レイヤをCXL独自のプロトコルにすることで実現
    - PCIe はGen 5.0以降は、PCIeのバス上で異なるプロトコルの通信を許容
- CXLは以下3種類のプロトコルの組み合わせで実現
  - CXL.io ：CXLデバイスの検知・エラー報告などに使用
  - CXL.cache ：アクセラレータがホストメモリ上のデータをキャッシュをする際に使用
  - CXL.mem ：CPUがアクセラレータ上メモリ/CXLメモリデバイスにRead/Writeする際に使用
- 以下の３種類のデバイスが定義

キャッシュを持ち
メモリを内蔵しない or
ホストに内蔵メモリを見せないデバイス

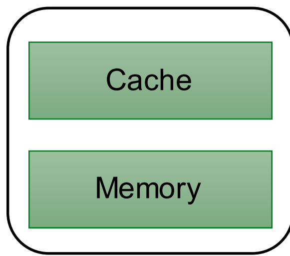


Type-1

(例：SMART NICや FPGAで該当する構成の場合)

使用するCXLプロトコル
- CXL.io
- CXL.cache

デバイスにキャッシュとメモリを内蔵するデバイス

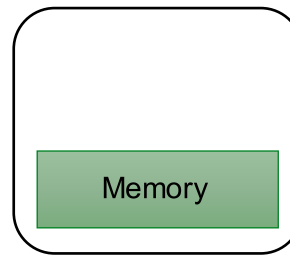


Type-2

(例：GPGPU、FPGAでメモリを内蔵するケースなど)

使用するCXLプロトコル
- CXL.io
- CXL.cache
- CXL.mem

CXLに接続する外付けの拡張メモリ
(揮発性でも不揮発性でもOK)



Type-3

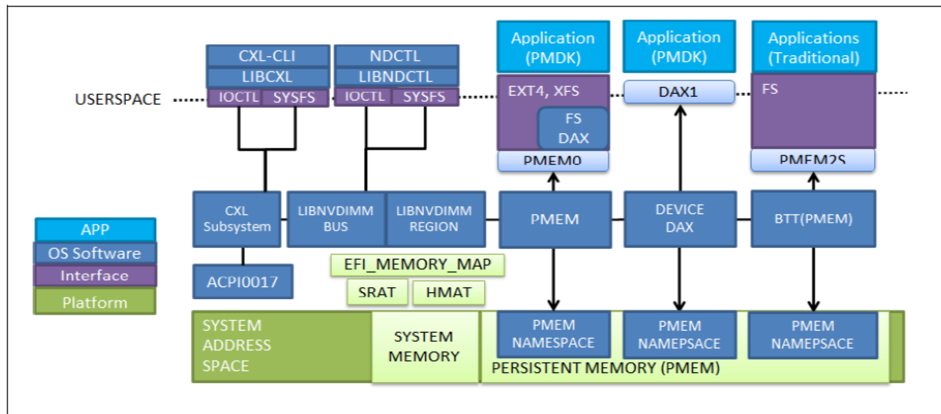
使用するCXLプロトコル
- CXL.io
- CXL.mem

© 2022 Fujitsu Limited

Type 3のメモリデバイスについて(1/2)

FUJITSU

- CXL Type3デバイスは揮発メモリや不揮発メモリを搭載可能
  - 注）“DIMM” Slotに挿すものではないので、CXLの不揮発メモリをNVDIMM(Non Volatile DIMM)とは呼べなさそう
    - とりあえず、この資料中でNVDIMMと明に記載している場合、CXLではなくDIMM slotに挿すほうの不揮発メモリを指すことにする
  - 不揮発メモリはCXL仕様のv2.0で対応
- 不揮発メモリのドライバも含めてCXL用に新規に作成が必要
  - ハードの認識・制御方法がNVDIMMとは大きく変わる
  - Linux communityではまだまだドライバの開発中の状態
- 一方従来のNVDIMMのregion/namespaceと似たMemory poolやnamespaceのコンセプトは継続
  - このCXL対応もcommunityで開発中の状態
    ⇒ 言い換えると、基本的な機能の開発もまだまだということ
- また、MW/アプリ向けのIF（右図赤枠内）も従来から継続のため、NVDIMMで利用できたMW/アプリはそのまま利用可能
  - Storage, Filesystem DAX, Device DAX
  - PMDK
- CXLはv2.0以降 Hotplugも対応
  - ということは、揮発メモリのHotplug対応デバイスが今後出てくると、LinuxのMemory Hotplug機構が動作する必要がある？
    - PRIMEQUESTのために作ったMemory Hotplug機能が、コンシューマー向けのマシンで動作する日がそのうち来るかも？



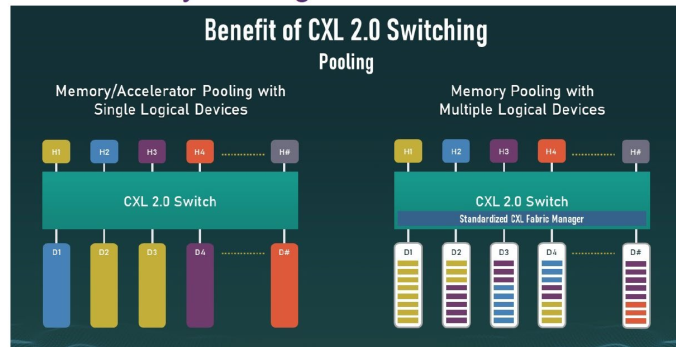
© 2022 Fujitsu Limited

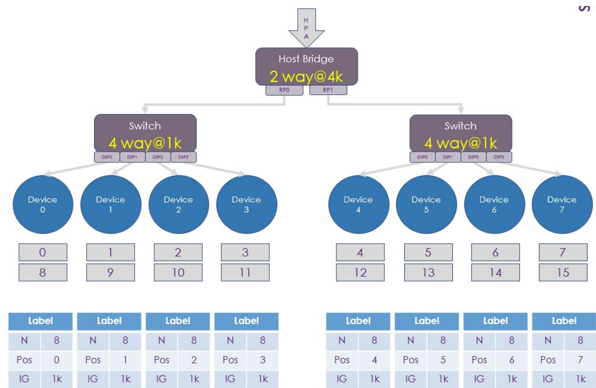
Type 3のメモリデバイスについて(2/2)

FUJITSU

- 以下についてはNVDIMMに引き続き設定可能となっている
  - CXLのメモリデバイスはプールとして分割して利用可能
    - 一つのメモリデバイスをそのまま一つのメモリ領域として提供してもよい
    - 複数のデバイスを束ねて1つのメモリ領域として見せてもよい
    - 一つのデバイスを複数の領域に見せることも可能
  - Interleaveの設定が可能
    - 右下はメモリデバイスを8wayのinterleave設定をした例
      - (図中にHost bridgeとかSwitchといった言葉が見られるが、前提知識が必要なので後述)

CXL 2.0 Memory Pooling





© 2022 Fujitsu Limited

CXLを使った製品やデモ、予定など

FUJITSU

- Intelは次期サーバ向けCPU Sapphire RapidsでCXL 1.1(*)をサポート
  (*)不揮発メモリやhotplugなどはまだ未対応の版
  https://cloud.watch.impress.co.jp/img/clw/docs/1345/040/html/005_o.jpg.html
- AMDも次期CPU Genoa/BergamoでCXL 1.1をサポート
  https://northwood.blog.fc2.com/blog-entry-11142.html
- ArmはNeoverse N2(Arm v9)でCXL2.0をサポート
  https://cloud.watch.impress.co.jp/docs/news/1321510.html
- SamsungはHPC向けにCXLのメモリデバイスを2021/5に発表
  - https://news.samsung.com/global/samsung-unveils-industry-first-memory-module-incorporating-new-cxl-interconnect-standard
  - 合わせて Scalable Memory Development Kit(SMDK)も発表している
    - 現在は特定の顧客にのみ公開
    - SMDKにより、「CXL memoryがAI, 機械学習、5G-edgeマーケットのためのデータセンター開発者がCXLmemoryによりアクセスしやすくなる」ととしている。
    - https://news.samsung.com/global/samsung-introduces-industrys-first-open-source-software-solution-for-cxl-memory-platform
- 2021/11/14~19に行われたスパコン関連のイベントSC21にて複数の企業がCXLのデモを行っている*
  - メモリデバイスのプールのデモ
  - CXLの接続(link up)、設定のread/write、メモリデバイスのread/write
  - FPGAベースのメモリ拡張コントローラのデモ
  - …など多数

* https://b373eaf2-67af-4a29-b28c-3aae9e644f30.filesusr.com/ugd/0c1418_4e1be90162be492a992e3fc485663915.pdf

© 2022 Fujitsu Limited

FUJITSU

CXLの仕様書を理解するための前提知識

～より深く知るために～

- PCI
- PCI express
- ACPI

© 2022 Fujitsu Limited

PCI

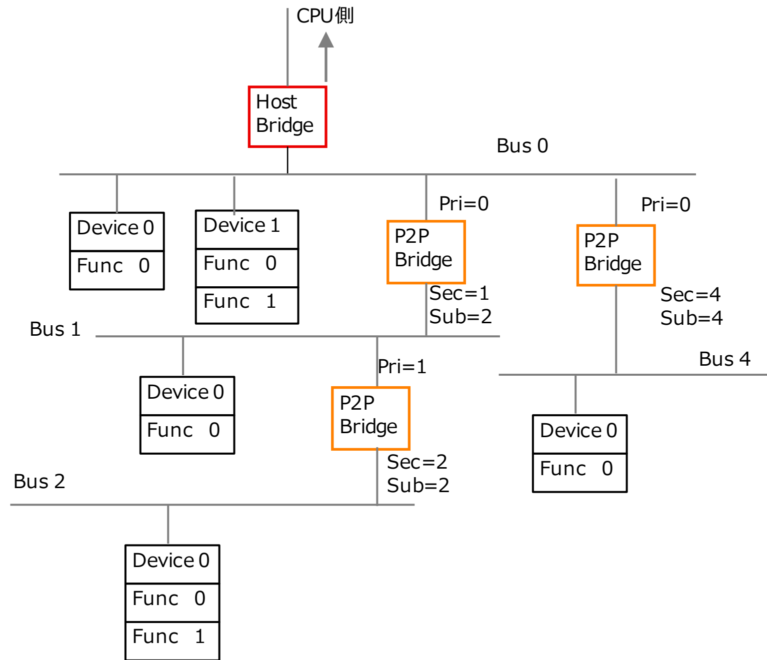
FUJITSU

- PCI (Peripheral Component Interconnect)
  - 直訳すると、「周辺装置のコンポーネントの内部接続」
    - つまり、ハードの内部接続のための規格
- データを送る信号がパラレル接続
  - 昔はパラレルで並列に信号を送ったほうが早かった
    - 現在身近なUSBもPCIeも今はシリアル接続だが、昔はそういう時代があった
  - デバイス間で信号線を“共有”するのでPCI “Bus”と呼ばれる
    - 余談だが、Busの語源は実は自動車のBusと同じで、ラテン語のomnibus(乗合馬車)
      - https://en.wikipedia.org/wiki/Bus_(computing)
    - デバイス間で誰が利用するのかを調停してBus線を使用する
- PCI expressやCXLの仕様のベースとして、以下のコンセプトは引き継がれている
  - Host bridgeやPCI-PCI(P2P) bridgeからなるツリー構造
  - Bus/Device/Function番号の概念
  - 認識、操作をするためのconfiguration空間

© 2022 Fujitsu Limited

Bridgeと木構造

- PCIはHost Bridgeを根とする木構造(右は説明のための図)
  - Host BridgeはChipsetのICH/South Bridgeに搭載されていた
  - Host Bridge直下のBusはBus番号＝０となる
  - PCI to PCI(P2P) ブリッジを介するとBus番号が変わる
    - P2P ブリッジは、親のBus番号(Primary)、下流のBusの番号の範囲を”Secondary番号～Subordinate番号”の間として覚えておく
    - アクセスが要求されたBusの番号が上流・下流それぞれ一致する場合に信号を通す
  - PCIの各スロットに挿すデバイスにDevice番号を割り当て
  - デバイスに複数の機能(マルチポートのEther NICなど)がある場合は、それぞれにFunction番号を割り当て
    - 機能が１つの場合はFunction番号０で固定
  - Bus番号・Device番号・Function番号の３つでコントロールしたいデバイスを指定する
    - ３つまとめてRequester IDと記載されることもある
    - 大型サーバでは複数Host Bridgeを持つこともでき、Bus番号の上にSegment番号を割り当てる
  - 番号やメモリマップする領域などのリソースの割り当て
    - システム起動時はfirmwareが実施
    - Hotadd時はOSが実施



- CXLの仕様書でも”PPB”(=p2p Bridge)といった記述で登場
- Device や Function、Requester IDという言葉はCXLの仕様書でも特に断りなく記載

© 2022 Fujitsu Limited

（参考）LinuxでのPCIの木構造の確認方法

FUJITSU

- lspci –vt コマンドでPCIの木構造を確認できる
  - QEMU/KvmのVM guestはHost bridge1つでp2p bridgeが無いので、比較的わかりやすい
    - 以下では、左から、Segment番号、bus番号、device番号、function番号の順に出力

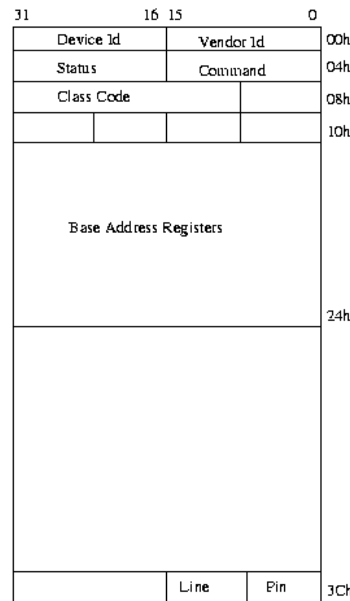
```
-[0000:00]-+-00.0  Intel Corporation 440FX - 82441FX PMC [Natoma]
           +-01.0  Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
           +-01.1  Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
           +-01.3  Intel Corporation 82371AB/EB/MB PIIX4 ACPI
           +-03.0  Red Hat, Inc. Virtio network device
           +-04.0  Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #1
           +-04.1  Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #2
           +-04.2  Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #3
           +-04.7  Intel Corporation 82801I (ICH9 Family) USB2 EHCI Controller #1
           +-05.0  Red Hat, Inc. Virtio console
           +-06.0  Red Hat, Inc. Virtio block device
           +-07.0  Red Hat, Inc. Virtio memory balloon
           ¥-08.0  Red Hat, Inc. Virtio RNG
```

- host bridge 1 つでP2P bridgeがないので、Segment 番号とBus番号は0 のみで[0000:00]
- Device番号は0,1,3,4,5,6,7,8が割り当てられており、複数の機能を持つものはFunction番号が割り当てられている

© 2022 Fujitsu Limited

PCIのConfiguration空間

- PCIデバイスを認識/操作するためのレジスタ空間
  - この空間へのアクセスは厳密にはアーキテクチャ依存
    - アクセス方法の違いを気にしなくて済むよう、Linuxでは例えば以下のようなAPIを使う
      - pci_read_config_word() : 指定したデバイス・レジスタの16bitの読み込み
      - pci_write_config_dword() : 指定したデバイス・レジスタの32bitの書き込み
- （Expressになる前の）PCIでは256byteのサイズ
  - このうち、0x0～0x3fはPCIの仕様として使い方が定められている
    - p2p bridge、Card bus, 通常のデバイスなど、フォーマット形式にいくつか種類がある
  - 0x40～0xffの残り192byteは、デバイス固有の操作のためにハードウェアベンダーが利用できる
- デバイスの認識、状態の取得、基本的な制御などを行える
  - 例
    - Vendor ID： デバイスを作成したベンダーの番号で、会社ごとに予約 （Fujitsuは4303（0x10CF)
      - ベンダーの一覧が掲載： https://pcisig.com/membership/member-companies
      - Intelは0x8086と洒落ている
    - Device ID: そのベンダーの中のデバイスを識別するための番号
    - Class Code: デバイスの種類、機能を表す番号
- PCIに接続するデバイスのドライバを書く人にとっては必須の知識
- PCIeやCXLでもこの基本構造は引き継がれている



configuration 空間の最初の共通部分

(出典) https://linuxjf.osdn.jp/JFdocs/The-Linux-Kernel-7.html

© 2022 Fujitsu Limited

PCI express

FUJITSU

- 現在はPCI expressがInterconnectの主流に
  - 最新仕様はGen 6.0が2022/1に公開
  - (CXLはGen 5.0で定められた仕様を利用)
- 旧PCIとの違い
  - Busの信号線がパラレルからシリアルに
  - データの渡し方が、TCP/IPのようなパケットになり、 3層のレイヤーが定義された
    - トランザクション層：デバイスやCPU間のデータ通信
    - データリンク層：エラーチェックやリトライ
    - 物理層：電気的な接続、初期化、リンクの確立など
  - configuration空間のサイズが4096byteに増えた（次ページ)
  - Root ComplexやSwitchによって各コンポーネントを接続するようになった(次ページ)

© 2022 Fujitsu Limited

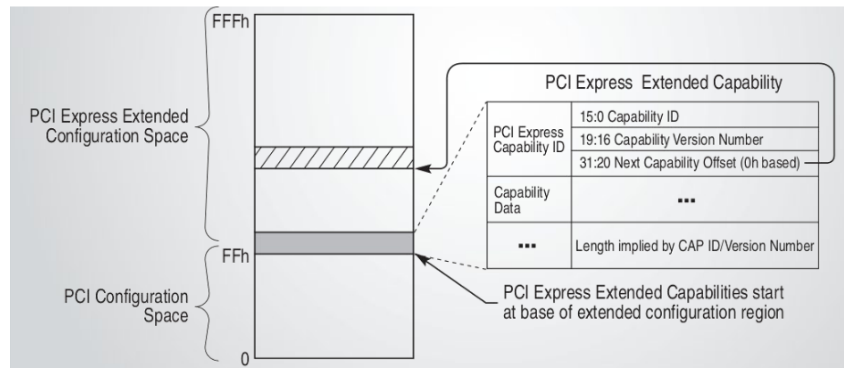
PCI expressのconfiguration空間

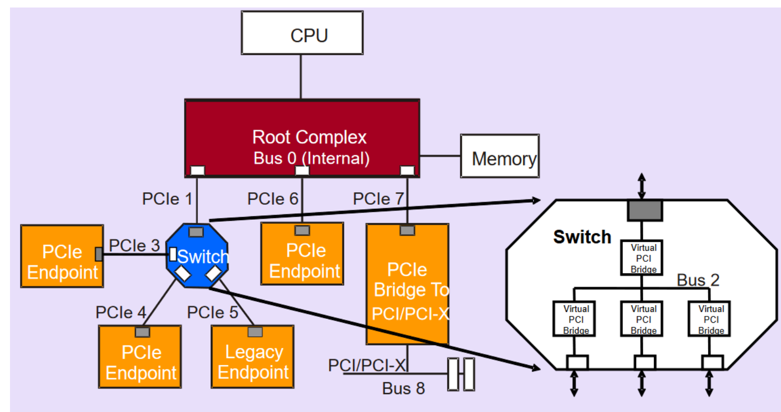
FUJITSU

- 4096byteにサイズが拡大
- 様々なExtended Capabilityが定義できるようになった(右上)
  - Extended Capabilityの中にはDVSEC(Designated Vendor-Specific Extended Capability)、すなわちベンダー固有の拡張が定義できる
    - ただし、LinuxのupstreamIに含まれているドライバでこのDVSEC capabilityを使っているものは少ない

      (図の出典): https://wiki.qemu.org/images/f/f6/PCIvsPCIe.pdf
  - CXLではPCIeのconfiguration空間のDVSECを利用している(後述)
- Root ComplexやSwitchによって各コンポーネントを接続（右下）
  - Root complexやSwitchの内部には(Virtual)PCI Bridgeが内包される

    (図の出典): https://pcisig.com/sites/default/files/files/PCI_Express_Basics_Background.pdf#page=26
    - 左図では別になっているが、今はCPUのダイの内部にRoot complexを持つ構成が多いと思われる





© 2022 Fujitsu Limited

ACPIとは

FUJITSU

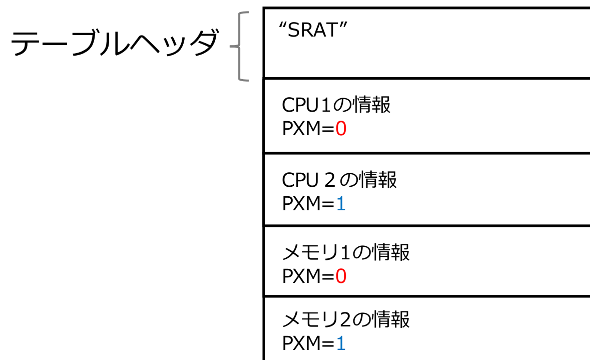
- ACPI (=Advanced Configuration and Power Interface)
  - もともとはPCの電源制御のための仕様
    - ACPIの登場以前はファームウェアだけで電源制御をしていた(APM)
      - OSが重要な処理を行っているときに、勝手に電源操作をされることがあり、トラブルのもとに
      - OSとファームが協力して電源操作を行うためにACPIを策定
  - 現在は電源だけでなくプラットフォーム全体のハード構成やその操作方法を表現
    - ファームとOS間のインタフェースとして重要な位置づけに
      - ハード構成の表現例
        - CPUとメモリの構成やアクセス性能
        - ハードの使用しているリソースの情報の取得
      - デバイスのHotplugもACPI経由で実行可能
  - x86が発祥だが、別にARMでも利用OK
    - 実際、富岳でもACPIを利用
      - 「スーパーコンピュータ「富岳」のシステムソフトウェアについて」
        - http://www.ipsj.or.jp/sig/os/index.php?plugin=attach&refer=ComSys2019&openfile=fugaku_system_software.pdf
  - NVDIMM向けのACPIの仕様も存在
  - CXLのための新仕様が新たにACPIに追加されてきている

© 2022 Fujitsu Limited

ACPIの仕様のポイント

FUJITSU

- 電源管理という観点ではCPUなどの状態(sleepなど)を定義
  - 今回はこちらについてはあまり触れない
- システム構成についての仕様は大きく分けると２種類(私見)
  - テーブル型(右上)
    - 起動時の「OSに制御が渡った直後」でも利用しやすい形式
      - 例）CPUとメモリの構成はOSのメモリ管理機構の初期化に必要
    - テーブルヘッダを先頭に、追加情報のテーブルを複数定義
    - 上図はSRAT(後述)と呼ばれるテーブルの例
  - ツリー型(右下)
    - システムの階層構造を表現できる
      - 例えば、Docking Stationのような、複数のハードの塊を丸ごとHotplugするといった機能を実現できる
        - ツリーの末端のデバイスから停止させて、最終的にその親ノードのデバイスを丸ごとEject
    - ハードの状態の取得、設定などをASLという独自コードで記述・表現する
      - コンパイルしてバイナリ形式にしたものがファームウェアに搭載
    - OSが利用するには、ACPIのドライバが動作開始し、このバイナリが解釈できる状態になっている必要がある
      - 右図はNode0にPCI Host Bridgeが存在する場合の定義例




```
Device(ND0) { // this is a node 0
  Name(_HID, "ACPI0004")
  // Returns the "Current Resources"
  Name(_CRS, ResourceTemplate() {
     …
   }
 )
 Device(PCI0) {
   Name(_HID, EISAID("PNP0A03"))
   Name(_ADR, 0x00000000)
   Name(_SEG, 0) // The buses below the host bridge belong to PCI segment 0
   …
 }
```


© 2022 Fujitsu Limited

(CXLではない)旧来のNVDIMMのためのACPI仕様

FUJITSU

- NFIT
  - NVDIMMの構成を表現し、OSが認識するためのテーブル型の定義
- NVDIMM Root Device
  - システム全体のNVDIMMについて、操作・情報取得などを行う
    - 識別のIDは“ACPI0012”
    - ツリー(DSDT/SSDT)の中で、システムボード(_SB)直下に定義する
    - システムにつき１つのみ定義
- NVDIMM Device
  - 個別のNVDIMMに対する捜査・情報取得用
  - 識別用のIDはなく、ツリー構造の中でNVDIMM Root Deviceの直下に定義される必要がある
    - System Board(SB)が１つで、その直下にNVDIMMを搭載することが前提の仕様(つまり、大型のサーバが複数SBを持つ場合を想定していない)

- _DSM (Device Specific Method)
  - デバイス固有機能の操作を提供するメソッド
    - OSのドライバがこのメソッドを実行することで動作
  - ツリー構造の中で定義
    - NVDIMM Root Device配下の_DSMの機能
      - Address Range Scrub 指定した範囲のメモリを舐めてエラー訂正の実行
      - Interleaveしている場合、システムの物理アドレスから、NVDIMMの特定とそのDIMM内アドレスへの変換
        - 筆者は故障したNVDIMMの特定のために、本機能を利用したコードをndctlコマンドに実装
      - etc..
    - NVDIMM Device配下の_DSM
      - S.M.A.R.T情報の取得
      - Error Injection
      - etc..

CXLでどうなったかは後述

© 2022 Fujitsu Limited

NVDIMMのASLの定義例

FUJITSU

```
Scope (¥_SB){
   Device (NVDR) // NVDIMM root device
   {
      Name (_HID, "ACPI0012")  //NVDIMM root deviceであることを示すID

      Method (_STA) {...}
      Method (_FIT) {...}
      Method (_DSM, ...) {    // NVDIMM root deviceの_DSMメソッド
                              // (ドライバがこれを評価すると、前述の機能が動作)
      ...
      }
      Device (NVD) // NVDIMM device
      {
         Name(_ADR, h) Method (_DSM, ...) { // NVDIMM deviceの_DSMメソッド (同上)
         ...
```


© 2022 Fujitsu Limited

そのほかの関連するACPIの仕様

FUJITSU

- SRAT
  - CPU/Memoryなどの組み合わせを知るためのテーブル
    - NUMA Nodeの構成をOSが起動時に理解するためのもの
      - proximity(PXM)の値が同じなら同一NUMA nodeに存在すると解釈
    - 現在は上記のほか、GPUなどのIO deviceについても扱う

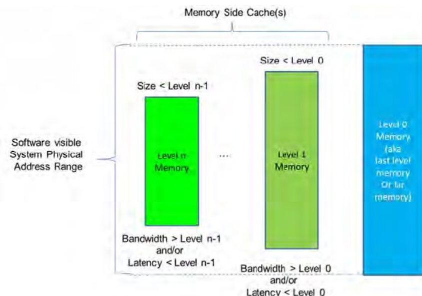
“SRAT”
CPU(*)1の情報 PXM=0
CPU 2 の情報 PXM=1
メモリ1の情報 PXM=0
メモリ2の情報 PXM=1

PXMの値でCPU1とメモリ１、CPU2とメモリ２がそれぞれ同じNUMA Nodeに属することがわかる

- SLIT
  - CPUからメモリへのアクセス速度を示すテーブル
    - proximityをindexとする行列となっている
    - 一番最速のメモリアクセス速度を10(=1.0倍)と定義
    - それに対して他のNode(PXM)は何倍ぐらいの距離(レイテンシー)になるかを表示
      - 右の例では、異なるNUMA Nodeへのレイテンシーは同じノードに対して2.1倍になる
    - 不揮発メモリ登場以前から存在

	0	1
0	10	21
1	21	10

- HMAT
  - SLITよりも正確なアクセス速度を示すテーブル
    - メモリのレイテンシー(pico秒)と帯域(MB/s)を表現
    - Intel DCPMMのMemory Modeのように、DRAMがPMEMのキャッシュになっているケースにも対応



© 2022 Fujitsu Limited

（参考）ACPIの内容の確認方法

FUJITSU

- 自分のLinuxマシンでどのようにACPIが定義・実装されているかは、実はだれでも確認できる
  - 実マシンだとかなり複雑でとっつきにくいため、初心者は構成が簡単なQEMU/KVMなどのVirtual Machineやクラウドのゲストから試してみるのがおすすめ

1. acpica-toolsパッケージをインストール
   または以下のサイトからソースをダウンロードしてビルド
   https://acpica.org/downloads

2. （必要ならデータ取得用のディレクトリを作成）
   # mkdir acpi_data; cd acpi_data

3. (root権限で)acpiのデータをバイナリで取得(コマンドを実行したカレントディレクトリにACPIのバイナリデータを出力
   # acpidump –b

4. <ファイル名.dat>のファイルを指定してiasl –dを実行すると、指定したファイル名対応するテーブル、ツリーのバイナリをディスアセンブルしたファイル<同じ名前.dsl>ができる
   # iasl –d <参照するテーブルの名前.dat>

© 2022 Fujitsu Limited

(参考)実際の出力例

FUJITSU

○ テーブル型(NFIT)の出力例
(先頭のみ)

```
/*
 * Intel ACPI Component Architecture
 * AML/ASL+ Disassembler version 20190509 (64-bit version)
 * Copyright (c) 2000 - 2019 Intel Corporation
 *
 * Disassembly of nfit.dat, Thu Jan 27 11:57:00 2022
 *
 * ACPI Data Table [NFIT]
 *
 * Format: [HexOffset DecimalOffset ByteLength]  FieldName : FieldValue
 */

[000h 0000   4]                    Signature : "NFIT"    [NVDIMM Firmware Interface Table]
[004h 0004   4]                 Table Length : 00000930
[008h 0008   1]                     Revision : 01
[009h 0009   1]                     Checksum : F1
[00Ah 0010   6]                       Oem ID : "FUJ   "
[010h 0016   8]                 Oem Table ID : "D3753-C1"
[018h 0024   4]                 Oem Revision : 00000002
[01Ch 0028   4]              Asl Compiler ID : "INTL"
[020h 0032   4]        Asl Compiler Revision : 20091013

[024h 0036   4]                     Reserved : 00000000

[028h 0040   2]                Subtable Type : 0004 [NVDIMM Control Region]
[02Ah 0042   2]                       Length : 0020

[02Ch 0044   2]                 Region Index : 0001
[02Eh 0046   2]                    Vendor Id : 8980
[030h 0048   2]                    Device Id : 0556
[032h 0050   2]                  Revision Id : 0000
[034h 0052   2]          Subsystem Vendor Id : 8980
[036h 0054   2]          Subsystem Device Id : 097A
[038h 0056   2]        Subsystem Revision Id : 0020
[03Ah 0058   1]                 Valid Fields : 01
[03Bh 0059   1]       Manufacturing Location : A2
:
```

○ ツリー型(DSDT)の出力例
(一部切り出し)

```
Mutex (OEML, 0x0F)
Device (PCI0)
{
    Name (_HID, EisaId ("PNP0A03") /* PCI Bus */)  // _HID: Hardware ID
    Name (_CID, EisaId ("PNP0A08") /* PCI Express Bus */)  // _CID: Compatible ID
    Name (_BBN, 0x00)  // _BBN: BIOS Bus Number
    Name (_ADR, 0x00)  // _ADR: Address
    OperationRegion (REGS, PCI_Config, 0x50, 0x30)
    Field (REGS, DWordAcc, NoLock, Preserve)
    {
        Offset (0x09),
        PAM0,   8,
        PAM1,   8,
        PAM2,   8,
        PAM3,   8,
        PAM4,   8,
        PAM5,   8,
        PAM6,   8,
        DRB0,   8,
        DRB1,   8,
        DRB2,   8,
        DRB3,   8,
        DRB4,   8,
        DRB5,   8,
        DRB6,   8,
        DRB7,   8,
            ,   6,
        HEN,    2,
        Offset (0x23),
        T_EN,   1,
        T_SZ,   2,
        Offset (0x2A),
        CRST,   1
    }
```


© 2022 Fujitsu Limited

FUJITSU

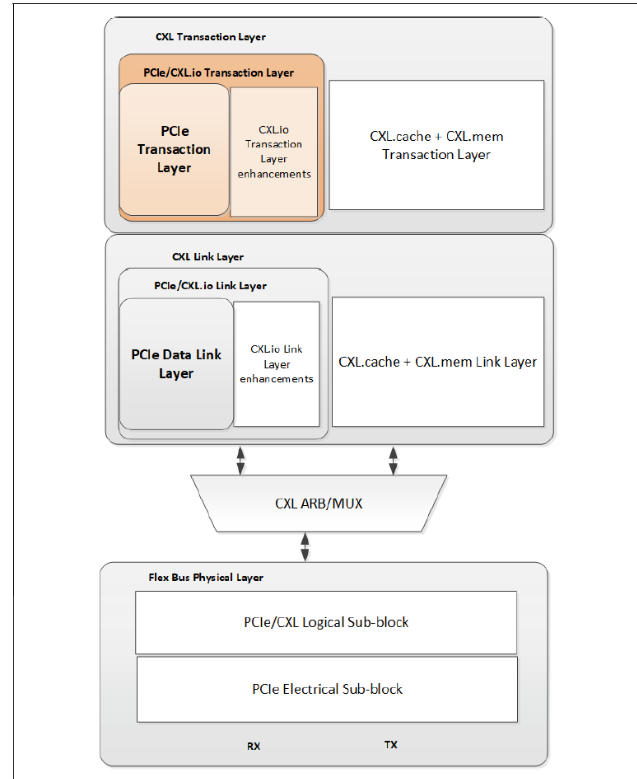
CXLの仕様

- レイヤー構造
- CXLのSwitchの機能
- configuration空間とデバイスの検出
- メッセージ

© 2022 Fujitsu Limited

CXLのレイヤー構造

- CXLもPCIeに習い３階層になっている
  - 前述のとおり、物理層はPCIeと共通
    - CXLのBusはFlex Busと記載
  - データリンク層とトランザクション層ではCXL.ioのプロトコルとCXL.cache，CXL.memの間でプロトコルが異なる。
    - CXL.ioはほぼPCIeのメッセージ形式を使ったプロトコルとなっている
      - type1,type2, type3の全てのデバイスで対応が必須
    - CXL.cacheとCXL.memのリンクレイヤーから上は独自形式のプロトコル
      - パケットの形式なども従来のPCIeとは異なる
  - Flex Busのポートは動作モードをCXL・PCIeどちらかに切り替えが可能



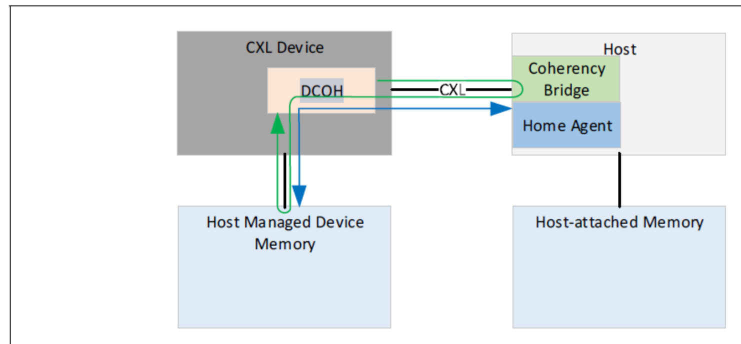
© 2022 Fujitsu Limited

Bias Based Coherency Model

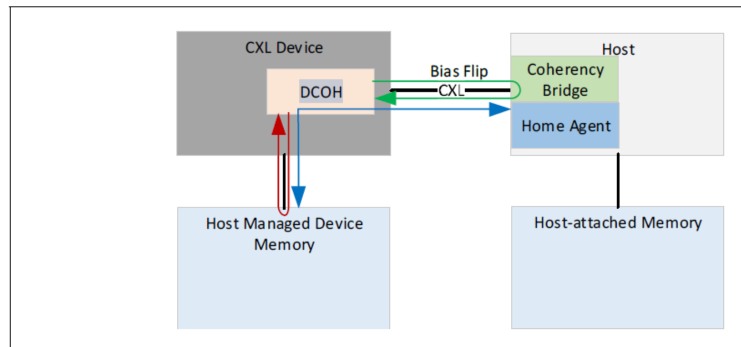
FUJITSU

- CXL Device側からそのDevice内(のHostに見せる)メモリへのアクセスには、以下の２種類の状態がある
  - Host Bias state（右上）
    - CPU側に接続されたメモリと同じく、キャッシュのcoherencyを保つためのリクエストをDevice側から出す必要がある
      - 図の緑の矢印のように、一旦CPU側にリクエストを出す必要がある
  - Device Bias state（右下）
    - CPU側がキャッシュラインを持っていないことが保証され、Device側は余計なリクエストを送らずにアクセスできる
      - Device Bias modeに切り替える（緑矢印）と、あとはデバイスが占有してアクセスできる（赤矢印）
- Device側のキャッシュの管理はDCOH(Device's Coherency engine)が担当
  - Device側のキャッシュの状態管理・更新と適切なメモリアクセス要求を行う必要がある

Type 2 Device - Host Bias



Type 2 Device - Device Bias

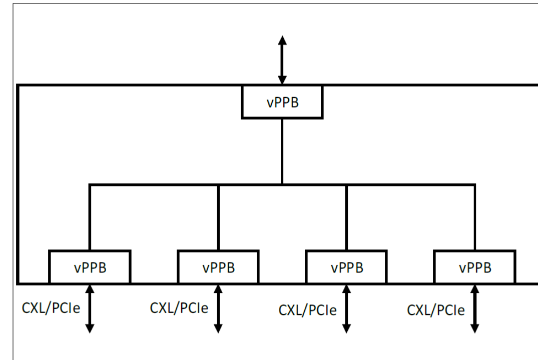


© 2022 Fujitsu Limited

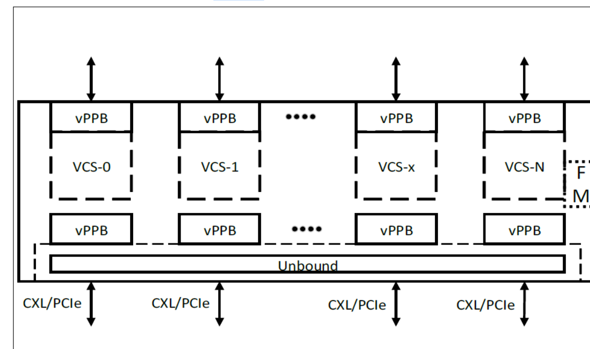
CXLのSwitch

- CXLのSwitchではupstream側のportが１つだけでなく、複数持つことが可能
  - 多対多の接続ができるようになる
    - CXL v2.0の仕様で定められたとのこと
    - 右上図はupstream側のport(virtual P2P bridge)が一つのケース
    - 右下のケースはupstream側のポートが複数搭載したケース
      - 内部的にはvirtual CXL Switch(VCS)が作られる

Example of a Single VCS Switch



Example of a Multiple VCS Switch with SLD Ports



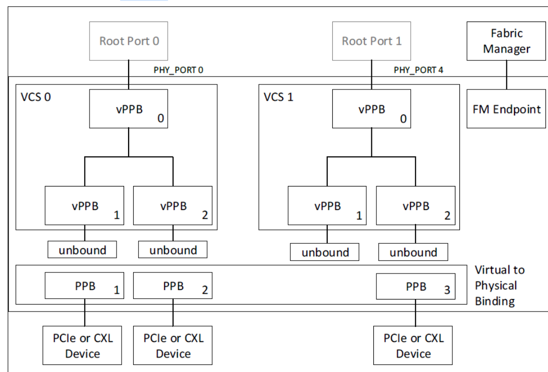
© 2022 Fujitsu Limited

Fabric Manager

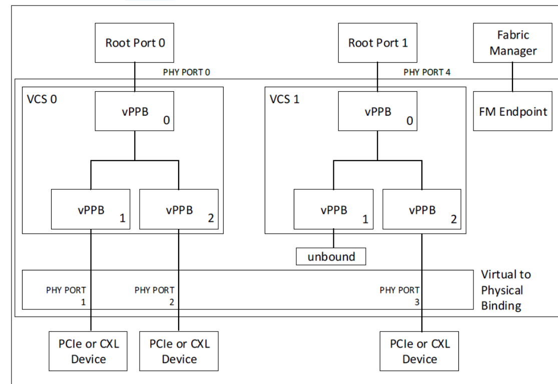
FUJITSU

- Downstream側のportについて、どこにbindするかを設定可能
  - Fabric Managerによって設定される
    - Fabric Managerの搭載はoptional
    - ただし、動的な構成をサポートする場合にはFabric Managerが必要となる
    - 右上はbind前、右下はbind後の状態となる
  - Fabric Managerをどのような形で実現するかについては自由であり、特に制約はないものとしている
    - ホスト上で動作するソフトであってもOK
    - BMCで動作する組み込みソフトでもOK
      - サーバ管理ソフトなどで実現する形
    - 他のCXLのDeviceやSwitch内の組み込みファームウェアでもOK
    - CXL Switch自身による"State Machine"として実装されてもOK

Example of CXL Switch Initialization When FM Boots First



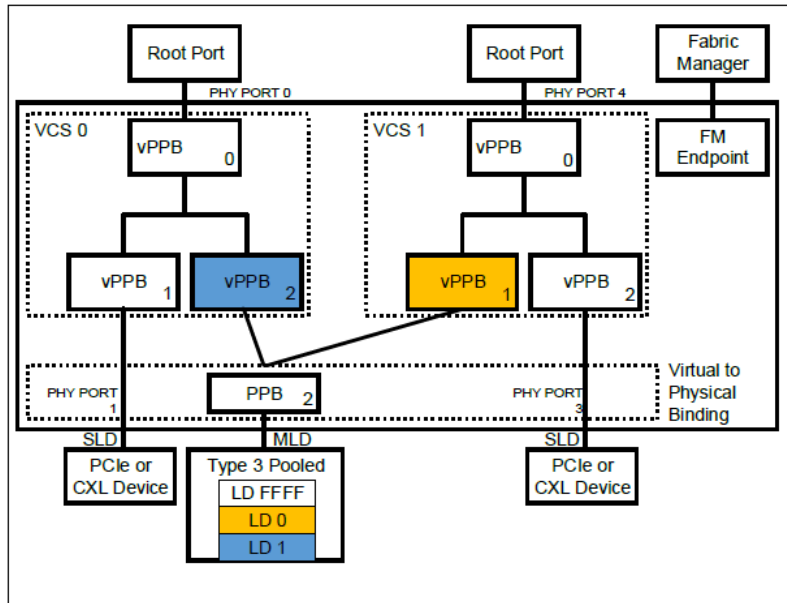
Example of CXL Switch after Initialization Completes



© 2022 Fujitsu Limited

Multi Logical Device

- Multi Logical Device(MLD)では、あるデバイスをあるvirtual P2P Bridge(vPPB)にbindし、もう片方は別のvPPBにbindするといった器用なことができる
  - 右図ではLD0とLD1をそれぞれ別のvPPBにbindしている
    - システム内だとメリットがあるかどうかわからないが、CPUから近いRoot Portから接続して性能を出したいとかの用途か？
    - システム外の同一ラックのサーバからの接続なら、他のシステムへ不揮発メモリを見せることが出来るようになるので、メリットは大きいかも



© 2022 Fujitsu Limited

Hotplug

FUJITSU

- CXL2.0ではhotplugに対応
  - HotAddと「手順を踏まえた」Hot removeはOK
    - Hot Remove時はハードのラッチを操作するか、ソフトで適切にeject操作を行う必要
      - ラッチが操作されると、OSに通知が渡りドライバによるeject操作が行われる
  - Surprising Hot Removeはgracefulにハンドリングする方法はないとしており、fatalな結果になることもある
    - 一応、そういう事象が起きたというビットはあるようで、ソフト側がそれをチェックして（エラーも含めた）何かしらの処理をおこなうことは可能な模様
- 全体の仕様としてはPCIeベースになっている模様で、あまり詳しくは書かれていない
  - “9.9 Hotplug”節の記述
    - “CXL leverages PCI Express Hot-plug model and Hot-plug elements as defined in PCI Express Specification and the applicable form factor specifications.”
- ただし、デバイス内にシステムに共有するメモリ(Host managed Device Memory:HDM)がある場合、その検出や設定をする必要がある
    - Root complexやUpstream側のSwitchにはHDM Decoder Capability Structureを実装
    - ソフト(OSのドライバ)がこれを利用
  - 詳しい手順については筆者もまだ学習中
    - 基本的には起動時と異なりファームウェアは介入せず、OSのドライバ側ですべて対処する必要があるはず。

© 2022 Fujitsu Limited

CXLのconfiguration空間

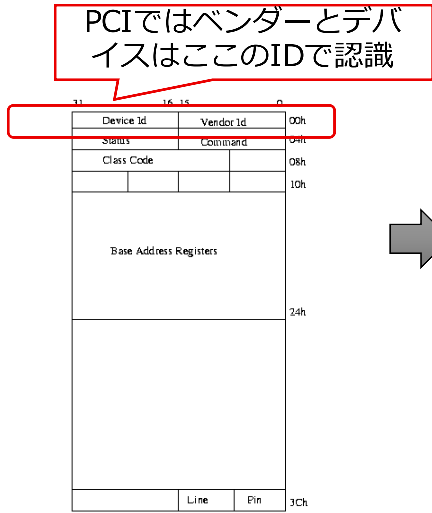
FUJITSU

© 2022 Fujitsu Limited

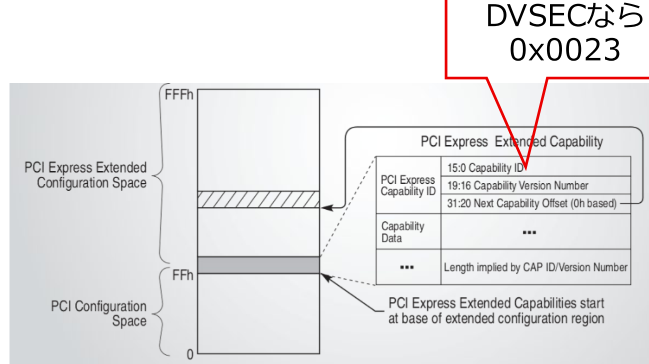
CXLのconfiguration空間

FUJITSU

○ CXLのコンフィグレーション空間はPCI expressの拡張CapabilityであるDVSEC(Designated Vendor-Specific Extended Capability)を使用しており、そのDVSEC Vendor IDで探索

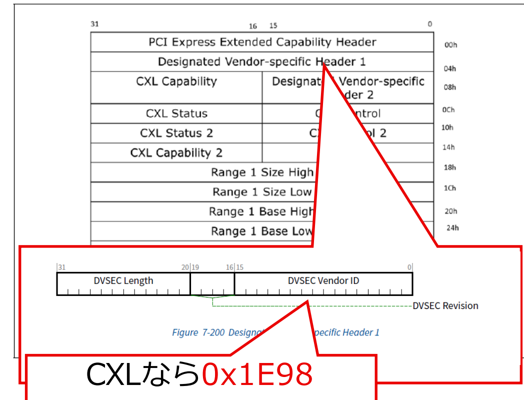


PCIのconfiguration 空間
(旧PCIとPCIe共通箇所)



- PCIに対してPCI expressは拡張configuration空間をオフセット0x100以降に持つことが出来る（上の図は上下逆なことに注意）
- Capability IDによって、色々なタイプの拡張Capabilityの形式が指定でき、DVSECはその一つ
- DVSECのCapability IDは0x0023

Figure 126. PCIe DVSEC for CXL Device



- DVSECの場合、Designated Vendor specific header 1の下位15ビットがDVSEC vender ID
- このDVSEC vendor IDが0x1E98だとCXLのconfiguration空間となる

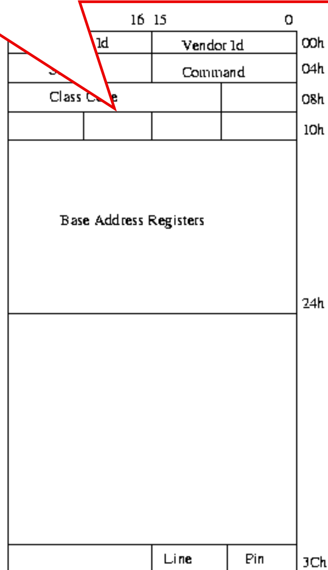
© 2022 Fujitsu Limited

Type 3 Memory Deviceのconfiguration空間

FUJITSU

○CXLのMemory Deviceは特別にPCIのクラスコードが割り当てられた

Class Codeに0x050210が入る



PCIのデバイスドライバーはこのクラスコードを見つけると
CXL memory device をprobeするものと思われる

- 左のように、(PCIeではなく)PCI(base)のconfig空間のところで判別できる
- PCIeのExtended Capabilityの中にはメモリデバイスのシリアル番号が入る
  （おそらく、不揮発メモリのデバイス識別用で、ホットプラグなどで、デバイスが入れ替わったりしたことを検出するため）
- Type1, Type2向けのクラスコードもあったほうがよさそうに思うが、ver2.0の仕様では定義されていない模様。

© 2022 Fujitsu Limited

upstreamのドライバ

FUJITSU

- この辺りはもっとも基本的な個所であるため、Linuxでもすでに前述の定義がドライバ上利用/実装済み

```
drivers/cxl/pci.h
---
/*
 * See section 8.1 Configuration Space Registers in the CXL 2.0
 * Specification
 */
#define PCI_DVSEC_HEADER1_LENGTH_MASK   GENMASK(31, 20)
#define PCI_DVSEC_VENDOR_ID_CXL         0x1E98
#define PCI_DVSEC_ID_CXL                0x0
----

drivers/cxl/pci.c
デバイスのprobe（認識）のところで使われている個所。
----
static int cxl_pci_probe(struct pci_dev *pdev, const struct pci_device_id *id)
{
            :
        rc = cxl_setup_regs(pdev, CXL_REGLOC_RBI_MEMDEV, &map);
            :


static int cxl_find_regblock(struct pci_dev *pdev, enum cxl_regloc_type type,
                             struct cxl_register_map *map)
{
        u32 regloc_size, regblocks;
        int regloc, i;

        regloc = pci_find_dvsec_capability(pdev, PCI_DVSEC_VENDOR_ID_CXL,
                                           PCI_DVSEC_ID_CXL_REGLOC_DVSEC_ID);
        if (!regloc)
                return -ENXIO;
```

```
drivers/cxl/pci.h
---
#define CXL_MEMORY_PROGIF     0x10
---

include/linux/pci_ids.h
---
#define PCI_BASE_CLASS_MEMORY          0x05
#define PCI_CLASS_MEMORY_RAM           0x0500
#define PCI_CLASS_MEMORY_FLASH         0x0501
#define PCI_CLASS_MEMORY_CXL           0x0502
#define PCI_CLASS_MEMORY_OTHER         0x0580
---

drivers/cxl/pci.c
---
static const struct pci_device_id cxl_mem_pci_tbl[] = {
        /* PCI class code for CXL.mem Type-3 Devices */
        { PCI_DEVICE_CLASS((PCI_CLASS_MEMORY_CXL << 8 | CXL_MEMORY_PROGIF), ~0)},


---
static struct pci_driver cxl_pci_driver = {
        .name                   = KBUILD_MODNAME,
        .id_table               = cxl_mem_pci_tbl,
        .probe                  = cxl_pci_probe,
        .driver = {
                .probe_type     = PROBE_PREFER_ASYNCHRONOUS,
        },
};
```


© 2022 Fujitsu Limited

CXLのCapabilityとコントロール

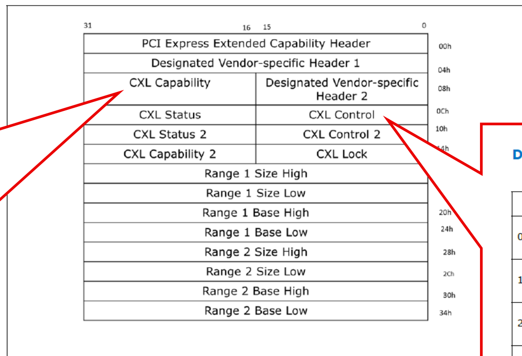
FUJITSU

- DVSECのCapabilityの領域から、どのprotocolが利用できて、実際に利用するのかなどの情報の取得、設定が可能

DVSEC CXL Capability (Offset 0Ah)

Bit	Attributes	Description
0	RO	Cache_Capable: If set, indicates CXL.cache protocol support when operating in Flex Bus.CXL mode. This must be 0 for all functions of an MLD.
1	RO	IO_Capable: If set, indicates CXL.io protocol support when operating in Flex Bus.CXL mode. Must be 1.
2	RO	Mem_Capable: If set, indicates CXL.mem protocol support when operating in Flex Bus.CXL mode. This must be 1 for all functions of an MLD.
3	RO	Mem_HwInit_Mode: If set, indicates this CXL.mem capable device initializes memory with assistance from hardware and firmware located on the device. If clear, indicates memory is initialized by host software such as device driver.

Figure 126. PCIe DVSEC for CXL Device



DVSEC CXL Control (Offset 0Ch)

Bit	Attributes	Description
0	RWL	Cache_Enable: When set, enables CXL.cache protocol operation when in Flex Bus.CXL mode. Locked by CONFIG_LOCK. Default value of this bit is 0.
1	RO	IO_Enable: When set, enables CXL.io protocol operation when in Flex Bus.CXL mode. This bit always returns 1.
2	RWL	Mem_Enable: When set, enables CXL.mem protocol operation when in Flex Bus.CXL mode. Locked by CONFIG_LOCK. Default value of this bit is 0.
		Cache_SF_Coverage: Performance hint to the device. Locked by CONFIG_LOCK. 0x00: Indicates no Snoop Filter coverage on the Host

- 例えば、このデバイスが（ハードとして）cache, io, memのうち、どのprotocolが利用可能なのか？はここを参照することで判別できる。

- （当該のデバイスのドライバが）cache, io, memのうち、実際にどのprotocolを使うのかを設定する
- （ただし、ioは実装が必須のため、常にonの模様）

© 2022 Fujitsu Limited

CXLのトランザクション層のメッセージ

© 2022 Fujitsu Limited

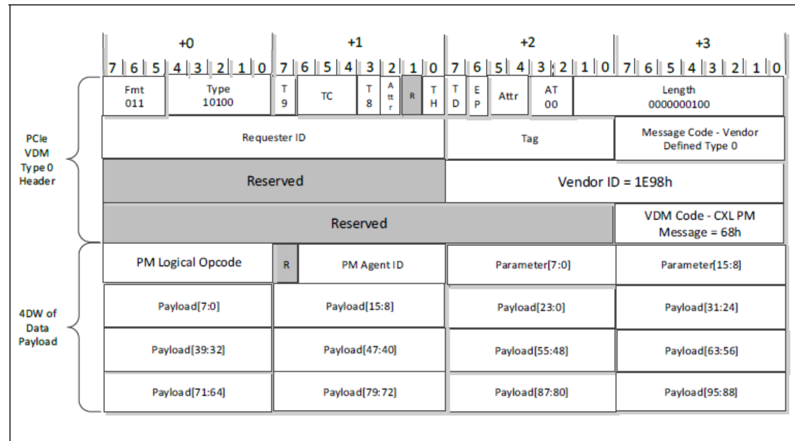
CXL.ioのプロトコル

FUJITSU

○ トランザクションレイヤーではPCIeのVendor Defined Messageを拡張したパケットになっている

- パケットの先頭はPCIeでのVendor Defined Messageのパケット構造（右図の上半分が相当）
  - Requester IDはbus番号、device番号、function番号を指定する
  - Byte12以降はVendorが独自に決められる
- CXLでは上記のVendor IDを0x1E98を指定してパケットを実行
  - 右図はpower managementのパケットで、この場合Codeに0x68を指定
  - それより下のオフセットでCXLの色々な参照、操作を行う

5. CXL Power Management Messages Packet Format



リンクレイヤーではCXL.ioのパケットはPCIeと同じ形式

© 2022 Fujitsu Limited

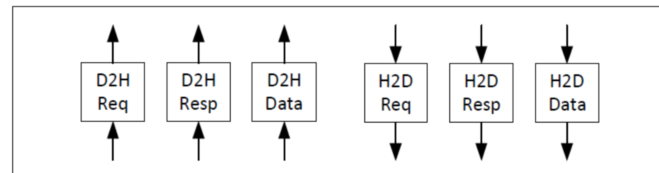
CXL.cacheのトランザクションレイヤープロトコル

FUJITSU

- ○前述のとおり、トランザクション層のプロトコルはCXL独自
  - ○CXL.cacheでは大きく６つのchannelを用意
    - ○HostとDevice間の向きで２パターン
    - ○request, response, dataの３パターン

      ⇒２×３＝６パターン
    - ○channelごとにフォーマットが策定されている
      - ● 右下はDeviceからHostへのrequestのフォーマット
        - 詳細については省略

CXL.cache Channels



注) CXL.cacheおよびCXL.memのデータリンク層のパケット形式もPCIeのそれとは異なる

本書では説明を省略する
興味のある人は、各自調べてみてほしい

CXL.cache - D2H Request Fields

D2H Request	Width	Description
Valid	1	The request is valid.
Opcode	5	The opcode specifies the operation of the request. Details in Table 18
Address [51:6]	46	Carries the physical address of coherent requests.
CQID	12	Command Queue ID: The CQID field contains the ID of the tracker entry that is associated with the request. When the response and data is returned for this request, the CQID is sent in the response or data message indicating to the device which tracker entry originated this request. Implementation Note: CQID usage depends on the round-trip transaction latency and desired bandwidth. To saturate link bandwidth for a x16 link @32GT/s, 11 bits of CQID should be sufficient.
NT	1	For cacheable reads the NonTemporal field is used as a hint to indicate to the Host how it should be cached. Details in Table 10
RSVD	14	
Total	**79**	

© 2022 Fujitsu Limited

CXL.memのトランザクション層のプロトコル

FUJITSU

- CPUのcoherent engineをMaster、メモリを持つデバイスをSubordinateと表現
  - デバイスがType 2(Accelerator)の場合は DCOHが存在し、キャッシュの調停ができるものとして扱う
- 以下の４つのメッセージを持つ
  - M2S（MasterからSubordinate)
    - Request without data (Req) (右はこのフォーマット)
      - CPUからデータの読み出しやキャッシュのinvalidateなどを要求するときに使う
    - Request with Data (RwD)
      - CPUからデータの書き込みを要求するときなどに使う
  - S2M(SubordinateからMaster)
    - Response without data (NDR:No Data Response)
      - Requestに対して応答を返すときに使う。(writeの結果とか)
    - Response with data (DRS:Data Response)
      - ReadのようなRequestに対してデータも返すときに使う
  - MemOpcodeのところでキャッシュのInvalidateやReadなど実行したい操作を指定

M2S Request Fields

Field	Bits	Description
Valid	1	The valid signal indicates that this is a valid request
MemOpcode	4	Memory Operation – This specifies which, if any, operation needs to be performed on the data and associated information. Details in Table 30
MetaField	2	Meta Data Field – Up to 3 Meta Data Fields can be addressed. This specifies which, if any, Meta Data Field needs to be updated. Details of Meta Data Field in Table 31. If the Subordinate does not support memory with Meta Data, this field will still be used by the DCOH for interpreting Host commands as described in Table 32
MetaValue	2	Meta Data Value - When MetaField is not No-Op, this specifies the value the field needs to be updated to. Details in Table 32. If the Subordinate does not support memory with Meta Data, this field will still be used by the device coherence engine for interpreting Host commands as described in Table 32
SnpType	3	Snoop Type - This specifies what snoop type, if any, needs to be issued by the DCOH and the minimum coherency state required by the Host. Details in Table 33
Address[51:5]	47	This field specifies the Host Physical Address associated with the MemOpcode. Addr[5] is provisioned for future usages such as critical chunk first.
Tag	16	The Tag field is used to specify the source entry in the Master which is pre-allocated for the duration of the CXL.mem transaction. This value needs to be reflected with the response from the Subordinate so the response can be routed appropriately. The exceptions are the MemRdFwd and MemWrFwd opcodes as described in Table 30
TC	2	Traffic Class - This can be used by the Master to specify the Quality of Service associated with the request. This is reserved for future usage.
LD-ID[3:0]	4	Logical Device Identifier - This identifies a logical device within a multiple-logical device.
RSVD	6	Reserved
Total	**87**	

© 2022 Fujitsu Limited

cache(coherency)の制御

FUJITSU

- CXL.cacheでは、Hostからのレスポンスなどでキャッシュの状態が示される
  - 右上の応答FieldのRspDataにMESIの情報(右下)が示される
- cxl.memプロトコルでも、データのRead/Writeの際にキャッシュの状態を更新したい場合にMetaStateとして指定できる
  - キャッシュとそのcoherent engine(DCOH)とメモリを持つType2デバイス向け
    - DCOHをデバイス側に持たないType 3デバイスにはこの指定は非適応
      - Type 3デバイスのキャッシュについてはHost側(CPU側)の問題とされている
- キャッシュをフラッシュするためのGPF(Global Persistent Flush)も定義されている
  - データが不揮発メモリに保存されたことを保証するための機能
  - NVDIMMと同じ用途だと思われる

H2D Response

CXL.cache - H2D Response Fields

H2D Response	Width	Description
Valid	1	The Valid field indicates that this is a valid response to the device.
Opcode	4	The Opcode field indicates the type of the response being sent. Details in Table 23
RspData	12	The response Opcode determines how the RspData field is interpreted as shown in Table 23. Thus, depending on Opcode, it can either contain the UQID or the MESI information in bits [3:0] as shown in Table 16.
RSP_PRE	2	RSP_PRE carries performance monitoring information. Details in Table 15
CQID	12	Command Queue ID: This is a reflection of the CQID sent with the D2H Request and indicates which device entry is the target of the response.
RSVD	1	
Total	**32**	

Cache State Encoding for H2D Response

Cache State	Encoding
Invalid (I)	4’b0011
Shared (S)	4’b0001
Exclusive (E)	4’b0010
Modified (M)	4’b0110
Error (Err)	4’b0100

© 2022 Fujitsu Limited

CXL向けのACPIの新仕様

© 2022 Fujitsu Limited

CXLに対するACPIの仕様追加

FUJITSU

- CXL Host Bridgeを見つけるためACPIのテーブルが用意
  - CEDT: CXL Host bridgeを見つけるのに利用するテーブル
    - 仕様の内容はACPIの仕様書ではなくCXLの仕様書(9.14 CXL OS Firmware Interface Extensions)に記載
    - Host bridgeのUnique IDによる認識、CXL1.1 or 2.0対応レベルの判別など
- ACPI Device ID (HID:Host Interface ID)が２つ割り当てられた
  - ACPIのデバイスツリー(DSDT/SSDT)上にCXLのデバイスを表現
    - ACPI0016: CXL Host bridge
    - ACPI0017: CXL Root Object (ただし、ACPIの正式仕様としてはまだ未公開)
- _CBR(CXL Host Bridge Register Info)メソッドが追加
  - 上記のACPI0016のデバイスに対して定義する。ドライバがこのメソッドを評価すると、Host BridgeのRegisterの場所や対応バージョン(CXL1.1 or 2.0)を取得できる

© 2022 Fujitsu Limited

例

○CXLのHost bridgeをACPIのASLで表現した例

```
Device(CXL0 ) {
  Name(_HID, EISAID ("ACPI0016")) // New HID to indicate CXL hierarchy
  Name(_CID, EISAID ("PNP0A08")) // To support legacy OSs that understands PCIe
  // but not the new HID
  Name(_UID, 0)
  Method (_CBR, 0) {
    Return( 0x00, DP_RCRB_BASE, 0x2000)
  }
  // Standard PCIe methods like _ BBN, _CRS.
  // PCIe _CRS describes .IO resources. PCIe _BBN describes bus number of CXL RCiEP
  // PCIe _OSC is used to negotiate control of CXL.IO capabilities
  ...
```


© 2022 Fujitsu Limited

CXLの関連仕様

- CDAT:Coherent Device Attribute Table
  - CXL Type 2のように、デバイス自身がメモリを持ち、かつCPU側のメモリについてもキャッシュコヒーレンシーについて責任を持つようになる
    ⇒ CPUだけでなく、デバイス側から見たメモリアクセス性能を取得する必要が出てきた
    - 従来のACPIのSLIT, HMATはCPUからのアクセス速度表現だった
    - CADTでは、アクセラレータからの各DIMMやアクセラレータ内へのメモリへのアクセス性能なども表現できる
      - 図ではS1,S2がCPU, ACC1～4がCXL上のアクセラレータデバイス
      - 例えば、ACC2->DIMM4へのメモリアクセス性能がどれくらいか？を表現可能

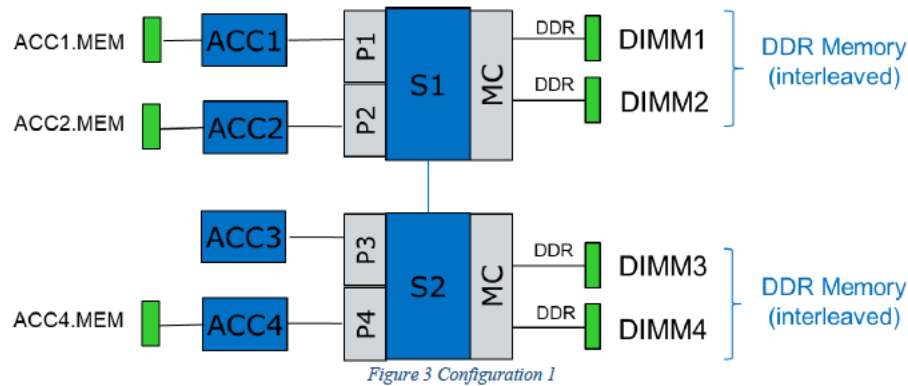


Figure 3 Configuration 1

- 現在のところ、独立した仕様として定義
  - CXL仕様本体に吸収されておらず、ACPIの仕様にもなっていない
    - 経緯は不明

© 2022 Fujitsu Limited

CXLにおけるType.3デバイス

特に不揮発メモリの扱いや、NVDIMMとの違い

© 2022 Fujitsu Limited

CXL搭載不揮発で使われなくなったACPI仕様

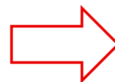
FUJITSU

- NFITと_DSM(*)はCXLでは使用されなくなった
  - NFITはテーブル型のため、CXLの木構造かつメモリをHotplugするような構成の表現にはやや不向き
    - 後からHotAddするケースのため、あらかじめ空白の領域を作っておく必要がある
  - NVDIMM Deviceに対する_DSMはベンダーごとの違いを吸収することを目指していたが、困難だった模様
    - 以下はIntelの_DSM仕様だが、当初はIntelに限らずNVDIMM全体の標準化を目指していた
    - しかし、今は仕様のタイトルにDCPMMと記載されており、DCPMMに限定された仕様になっている https://pmem.io/documents/NVDIMM_DSM_Interface-V1.8.pdf
    - かつてはNVDIMM-Nを出していたHPEも、独自の_DSMの仕様を提案していたが、上記とは多少異なる仕様となっていた https://github.com/HewlettPackard/hpe-nvm/blob/master/Documentation/NFIT_DSM_DDR4_NDVIMM-N_v88s.pdf

Learnings from ACPI Based Approach

Pros	Cons
▪ ACPI NFIT Unified NVDIMM-N and Intel's Optane PMem ▪ Helped get enabling upstream early ▪ _DSMs allowed a generic kernel implementation ▪ Differences abstracted away by _DSMs ▪ Mechanism has evolved gracefully ▪ Fairly small additions, few errata	▪ ACPI dynamic support doesn't scale ▪ Hot plug challenging ▪ Meant for small number of empty sockets ▪ _DSM complexity hard to maintain ▪ Bug fixes, additions logistical challenges ▪ Virtually impossible to support multiple devices ▪ No generic BIOS

(*) 厳密にはNVDIMM Root ModuleおよびNVDIMM Moduleに対する_DSM



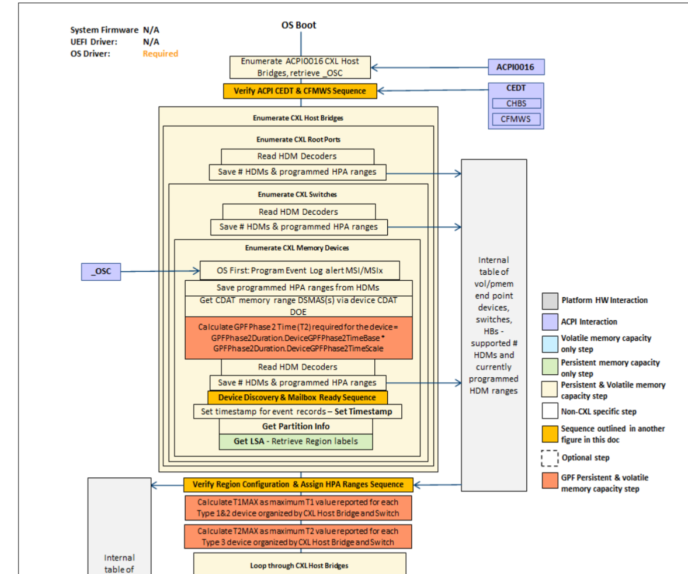
使わなくなった仕様に対する代替手段が必要

© 2022 Fujitsu Limited

CXL Type3メモリの認識

- 起動時に認識できるようにCEDTに仕様が追加
  - CXL Host bridgeの情報のほかに、メモリ領域読み取りのための情報が追加
  ⇒ CFMWS(CXL Fixed Memory Window Structure)
    - ただし、2022/2現在、まだEvaluation Copyとなっており、仕様本体にマージされていない
- CXL上のメモリの見つけ方（など）について、Intelがガイドラインを策定
  - “CXL* Type 3 Memory Device Software Guide”
    https://cdrdv2.intel.com/v1/dl/getContent/643805?wapkw=CXL%20memory%20device%20sw%20guide
    - CEDTのCFMWSをはじめ、いろいろな仕様の設定や読み取りに手順について、ファームウェア、EFI、OSの役割分担や順序がきっちり書かれている
      - かつてのACPIの仕様の記述では、「互いの役割分担がよくわからない」という声があり、OSのドライバやプラットフォームの開発者にとっては有益
  - ざっくりとした分担(システム起動時)
    - ファームウェア側
      - 必要なACPI仕様の用意
        - CEDT, SRAT, HMAT, ACPI0016, ACPI0017...
    - EFIやOS側
      - 上記をもとにCXL Host bridgeからSwitchをたどりMemory Deviceを検出し、メモリとしての領域や設定を検出
  - システム起動時の認識方法のほか、Hot Add、Hot Remove時の手順も記載

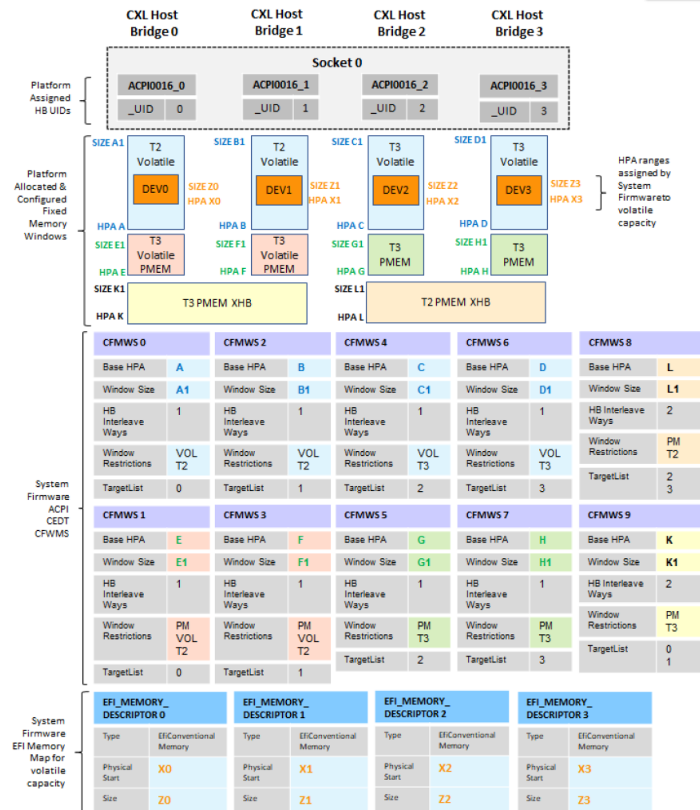
Figure 36 - High-level sequence: OS boot



© 2022 Fujitsu Limited

InterleaveとCEDT CFMWS

- メモリのInterleaveの設定はCXL Host BridgeやCXL Switchが担当
  - システム起動時は、ファームがこの設定を読み取りCEDTのCFMWSの内容に反映し、EFIやOSがそれを読み取る必要
    - CEDTのCFMWSの内容
      - ベースとなるホスト上での物理アドレス(HPA)
      - Window size(この領域のサイズ)
      - Interleaveに使われるMemory Deviceのway数
      - このHost bridgeにおけるGranularity(分割サイズ)
      - Window restriction（この領域の制約）
        - volatile or persistent, type2 or type3, ... etc
    - 右図はCFMWSの構成例
      - CXL Host Bridge 0～3の下に、Type2やType3の揮発メモリ、またインターリーブされたType2やType3の不揮発メモリを組み合わせて、計10個の領域を見せている
        - 図中の背景色や文字の色がそれぞれ対応



© 2022 Fujitsu Limited

CXLにおけるType 3デバイスの認識や操作

FUJITSU

- __DSMのようにACPI経由で操作をするのではなく、デバイスを直接コマンドで操作
  - 言い換えると、メモリマップされた領域を直接読み書きすることでコマンドを実行することになる
- ただし、_DSMにあったいくつかの機能は標準化（右）
  - Health情報の取得
  - Error Injection
  - など
- 右表のRequired列のMはMandatory, PMはPersistent Memoryの場合Mandatory、OはOptional

CXL Memory Device Command Opcodes

Opcode					Required	Input Payload Size (B)	Output Payload Size (B)
Command Set Bits[15:8]		Command Bits[7:0]		Combined Opcode			
40h	Identify	00h	Identify Memory Device (Section 8.2.9.5.1.1)	4000h	M	0	43h
41h	Capacity Config and Label Storage	00h	Get Partition Info (Section 8.2.9.5.2.1)	4100h	O	0	20h
		01h	Set Partition Info (Section 8.2.9.5.2.2)	4101h	O	0Ah	0
		02h	Get LSA (Section 8.2.9.5.2.3)	4102h	PM	8	0+
		03h	Set LSA (Section 8.2.9.5.2.4)	4103h	PM	8+	0
42h	Health Info and Alerts	00h	Get Health Info (Section 8.2.9.5.3.1)	4200h	M	0	12h
		01h	Get Alert Configuration (Section 8.2.9.5.3.2)	4201h	M	0	10h
		02h	Set Alert Configuration (Section 8.2.9.5.3.3)	4202h	M	0Ch	0
		03h	Get Shutdown State (Section 8.2.9.5.3.4)	4203h	PM	0	1
		04h	Set Shutdown State (Section 8.2.9.5.3.5)	4204h	PM	1	0
43h	Media and Poison Mgmt	00h	Get Poison List (Section 8.2.9.5.4.1)	4300h	PM	10h	20h+
		01h	Inject Poison (Section 8.2.9.5.4.2)	4301h	O	8	0
		02h	Clear Poison (Section 8.2.9.5.4.3)	4302h	O	48h	0
		03h	Get Scan Media Capabilities (Section 8.2.9.5.4.4)	4303h	PM	10h	4
		04h	Scan Media (Section 8.2.9.5.4.5)	4304h	PM	11h	0
		05h	Get Scan Media Results (Section 8.2.9.5.4.6)	4305h	PM	0	20h+
		00h	Sanitize	4400h	O	0	0

© 2022 Fujitsu Limited

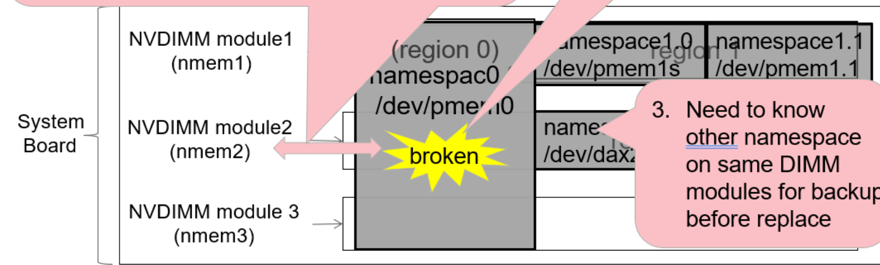
RAS:CXLで見当たらなくなった機能

- NVDIMMの_DSMでは存在したが、CXLの仕様中に見当たらない機能
  - ホストの物理アドレスから、デバイスを特定する機能
    - 故障したブロックからデバイスを特定して交換するため必要
    - NVDIMMではACPIの_DSMを使って特定できた
      - ndctlコマンドが_DSMを利用
      - CXLではこれがなくなった
    - CXLでどうするべきかは要検討か？
      - ハードのログに故障の記録が残っていればよい？
        - とはいえ、過去の経験ではソフト側が異常を検出しても、ハードにログが残ってないことが時々あったり….
      - ソフト側で代わりに計算して求める？
        - 求め方はまだ調査中
  - Address Range Scrub機能の起動
    - Scrub機能：メモリを舐めて一時的なエラーを訂正
    - NVDIMMでは_DSMでこの機能をStartさせることができた
    - CXLでは同様な機能の記述がない
      - この辺りは基本ベンダー任せか？

Replacing broken NVDIMM is complex(2/2)

FUJITSU

- To replace broken NVDIMM, the following information is necessary, at least



Copyright 2018 FUJITSU LIMITED

© 2022 Fujitsu Limited

まとめ

© 2022 Fujitsu Limited

個人的な所感

FUJITSU

- GPGPUをはじめとした将来のデバイスをつなぐ次世代のインターコネクトとして、インフラには重要な位置づけになる可能性
  - 分散システムとか、機械学習とか今後の様々なトレンド技術に対する縁の下の力持ちとして、PCIeにかわる必須のインターコネクトになるかも？
  - PCIeをベースにしているのも、筋としては悪くないという印象
- ただ、仕様もハードおよびソフトの実装も本格的になるのはこれからか？
  - 仕様自体が大きな目標を目指しており、実装が大変
    - Linuxのドライバの実装もまだまだこれから
      - IntelのLinux kernelの開発エンジニアはCXL対応で非常に忙しいらしく、最近はレスポンスが遅れ気味
    - QEMU/KVMがCXLをサポートするようになるのはさらに先と思われる
  - まだCXL本体にマージされていない仕様も存在
  - 今後ハード・ソフトが一旦実現されて、それで問題点が発見され、それが改善されてからが本番か？
- PC アーキテクチャ全体をまとめて教えてくれる入門書が必要
  - CPUだけでなく、PCI、ACPI、uEFIにCXLが追加され、関連仕様が多すぎて若者にはつらいと予想
    - と書くと「君が書くのを楽しみにしている」と煽てられるが、私だってこれを書くだけで一苦労する
- Qiita(ブログ)でCXLについて記述してくれたredvespidさんに感謝！

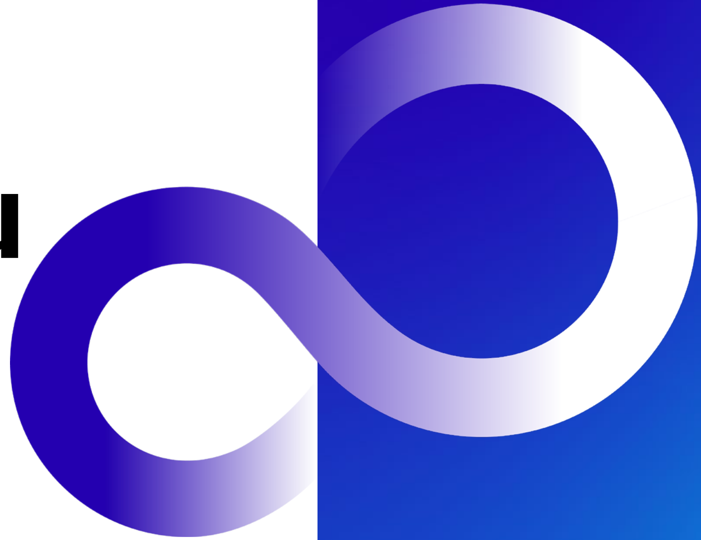
© 2022 Fujitsu Limited

参考文献

- 仕様書など
  - Compute Express Link Specification
    - https://www.computeexpresslink.org/download-the-specification
  - CXL* Type 3 Memory Device Software Guide
    - System Firmware, UEFI and OS Software Implementation Guide
      - https://cdrdv2.intel.com/v1/dl/getContent/643805?wapkw=CXL%20memory%20device%20sw%20guide
  - Coherent Device Attribute Table (CDAT) Specification
    - https://uefi.org/sites/default/files/resources/Coherent%20Device%20Attribute%20Table_1.02.pdf
- 海外カンファレンスでの発表資料
  - Compute Express Link™2.0: A High-Performance Interconnect for Memory Pooling
    - https://www.snia.org/educational-library/compute-express-link-20-high-performance-interconnect-memory-pooling-2021
  - Compute Express Link + Linux + QEMU = Yes
    - https://linuxplumbersconf.org/event/11/contributions/906/
- 日本語解説
  - redvespidさんによるCXLの解説
    - https://qiita.com/redvespid

© 2022 Fujitsu Limited

Thank you



FUJITSU

© 2022 Fujitsu Limited