ハイパーバイザの作り方〜ちゃんと理解する仮想化技術〜 第16回 PCI パススルーその2「VT-d の詳細」

はじめに

前回は、PCI パススルーと IOMMU の概要について解説しました。今回は、VT-d の詳細について解説していきます。

前回のおさらい

PCI デバイスが持つメモリ空間をゲストマシンのメモリ空間にマップすることにより PCI デバイスをパススルー接続できますが、DMA を使用するときに 1 つ困った問題が生じます。

PCI デバイスは DMA 時のアドレス指定にホスト物理アドレスを使用します。ゲスト OS は、ゲスト OS は自分が持つゲスト物理ページとホスト物理ページの対応情報を持っていないので正しいページ番号をドライバに与えることができません。

そこで、物理メモリと *PCI* デバイスの間に *MMU* のような装置を置きアドレス変換を行う方法が考え出されました。このような装置を *IOMMU* と呼びます。DMA 転送時にアドレス変換を行うことで、パススルーデバイスが正しいページへデータを書き込めるようになります（図 1)。Intel VT-d は、このような機能を実現するためにチップセットへ搭載された IOMMU です。

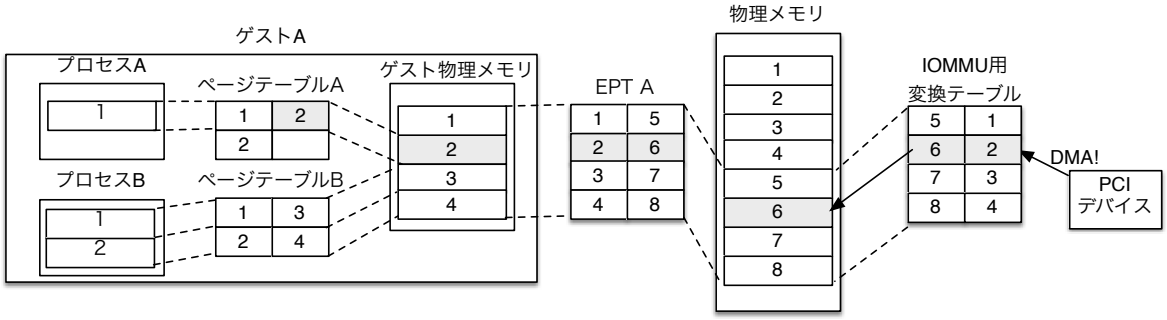

図 1　IOMMU を用いた DMA 時のアドレス変換

VT-d の提供する機能

VT-d が提供する機能には、次のようなものが挙げられます。

- IO device assignment：特定のデバイスを特定の VM に割り当てるための機能
- DMA remapping：仮想マシン上へ DMA するためにアドレス変換を行う機能
- Interrupt remapping：特定のデバイスから特定の VM へ届くように割り込みをルーティングする機能
- Reliability: DMA・割り込みエラーをシステムソフトウェアに記録・レポートできる

今回は VT-d によるアドレス変換の話を解説することを目的としているため、このうち “DMA remapping” 機能に絞って解説を進めていきます[*1]。

なお、VT-d に関するより詳しい内容は “Intel Virtualization Technology for Directed I/O Architecture Specification” という資料にて解説されているので、こちらをご覧下さい[*2]。

アドレス変換テーブル

VT-d では、アドレスリマップ対象のデバイスごとに CPU の MMU と同様の多段ページテーブルを持ちます。デバイスごとのページテーブルを管理するため、PCI デバイスを一意に識別する Bus Number・Device Number・Function の識別子から対応するページテーブルを探すための 2 段のテーブルを用います。

1 段目は Root Table と呼ばれ、0 から 255 までの Bus ナンバーに対応するエントリからなるテーブルです。このテーブルはアドレス変換時に VT-d から参照するため、Root Table Address Register へセットされます。 Root Table エントリのフォーマットを表 1 に示します。

bits	field	description
127:64	reserved	予約フィールド
63:12	context-table pointer	Context-table のアドレス
11:1	reserved	予約フィールド
0	present	このエントリが有効化どうか

表 1: Root table entry format

Root table エントリは context-table pointer フィールドで 2 段目のテーブルである Context-table のアドレスを指します。Context-table は Root table エントリで示される Bus 上に存在する Device 0-31・Function

[*1] より正確には、DMA remapping 機能のうち “Requests-without-PASID” であるものについてのみ解説しています。
[*2] http://www.intel.co.jp/content/www/jp/ja/intelligent-systems/intel-technology/vt-directed-io-spec.html

0-7 の各デバイスに対応するページテーブルを管理しています。Context-table エントリのフォーマットを表 2 に示します。

bits	field	description
127:88	reserved	予約フィールド
87:72	domain identifier	ドメイン ID このエントリがどの VM に属するかを示す
71	reserved	予約フィールド
70:67	ignored	無視されるフィールド
66:64	address width	ページテーブルの段数を示す (0x0:2 段、0x1:3 段、0x2:4 段、0x3:5 段、0x4:6 段)
63:12	second level page translation pointer	アドレス変換に使用するページテーブルエントリのアドレスを指定する
11:4	reserved	予約フィールド
3:2	translation type	アドレス変換時の挙動を設定
1	fault processing disable	0x0 :フォールトレコード・レポートを有効化 0x1 : フォールトレコード・レポートを無効化
0	present	このエントリが有効化どうか

表 2: Context-table entry format

Context-table エントリは second level page translation pointer でページテーブルのアドレスを指します。ページテーブルの段数は address width フィールドで指定されます。図 2 に 4 段ページテーブル・4KB ページを使用する場合のアドレス変換テーブルの全体図を示します。ここで使用されるページテーブルエントリのフォーマットは通常のページテーブルエントリと若干異なるのですが、ここでは解説は割愛します。

フォールト

変換対象になるアドレスに対する有効なページ割り当てが存在しない場合、または対象ページへのアクセス権がない場合、VT-d はフォールトを起こします。フォールトが発生した場合、メモリアクセスを行おうとした PCI デバイスはアクセスエラーを受け取ります。OS へは、MSI 割り込みを使用して通知されます。

IOTLB

IOMMU のアドレス変換を高速に行うには、通常の MMU と同じようにアドレス変換結果のキャッシュが必要です。通常の MMU ではこのような機構のことを TLB を呼びますが、IOMMU では IOTLB と呼びます。通常の MMU の TLB では、TLB エントリが古くなったときに invalidate と呼ばれる操作によりエント

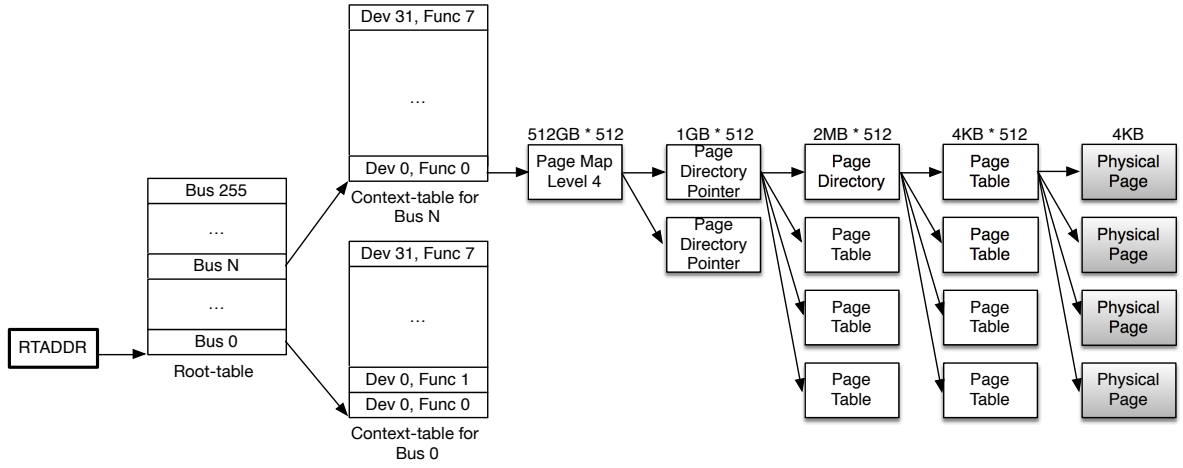

図 2　VT-d のアドレス変換テーブル全体図 (例)

リを削除します。このときの invalidate の粒度は、グローバルな invalidate・プロセス単位の invalidate([*3]:・ページ単位の invalidate などが選べます。VT-d の IOTLB では、グローバルな invalidate・デバイス単位の invalidate・VM 単位（ドメインと呼ばれる）の invalidate・ページ単位の invalidate が行えるようになっています。

Context-cache

IOTLB に類似していますが、VT-d では Context-table entry もキャッシュされています。これについても場合によって invalidate 操作が必要になります。

DMAR による IOMMU の通知

DMA リマッピング機能がハードウェア上に存在することを OS に伝えるため、ACPI は DMAR と呼ばれるテーブルを用意しています。DMAR では、いくつかの異なる種類の情報が列挙されています。

IOMMU は DMA Remapping Hardware Unit Definition(DRHD) という名前の構造体で記述されており、他に DMA に用いることのできない予約済みメモリ領域を示す Reserved Memory Region Reporting(RMRR) などが存在します。DRHD は IOMMU のレジスタベースアドレスと、IOMMU が DMA リマッピング対象にしている PCI デバイスのリストを持ちます。

VT-d の設定を OS 上から簡単に確認することは難しいですが、ACPI テーブルは簡単に見ることができるので、ここでその方法を説明します。例として Ubuntu Linux で DMAR を表示するコマンドを画面 1 に示します。

[*3] Tagged TLB の場合。

画面 1 DMAR **表示コマンド** (Ubuntu Linux)

```
$ sudo apt-get install iasl
$ sudo cp /sys/firmware/acpi/tables/DMAR .
$ sudo iasl -d DMAR
Intel ACPI Component Architecture
AML Disassembler version 20100528 [Dec 19 2012]
Copyright (c) 2000 - 2010 Intel Corporation
Supports ACPI Specification Revision 4.0a
Loading Acpi table from file DMAR
Acpi Data Table [DMAR] decoded, written to "DMAR.dsl"
syuu@hiratake:~$ cat DMAR.dsl
/*
* Intel ACPI Component Architecture
* AML Disassembler version 20100528
*
* Disassembly of DMAR, Mon Nov 25 09:11:42 2013
*
* ACPI Data Table [DMAR]
*
* Format: [HexOffset DecimalOffset ByteLength]  FieldName : FieldValue
*/
[000h 0000  4]                    Signature : "DMAR"    /* DMA Remapping table */
[004h 0004  4]                 Table Length : 00000130
[008h 0008  1]                     Revision : 01
[009h 0009  1]                     Checksum : 22
[00Ah 0010  6]                       Oem ID : "AMI"
[010h 0016  8]                 Oem Table ID : "OEMDMAR"
[018h 0024  4]                 Oem Revision : 00000001
[01Ch 0028  4]              Asl Compiler ID : "MSFT"
[020h 0032  4]        Asl Compiler Revision : 00000097
[024h 0036  1]           Host Address Width : 26
[025h 0037  1]                        Flags : 01
[030h 0048  2]                Subtable Type : 0000 <Hardware Unit Definition>
[032h 0050  2]                       Length : 0020
[034h 0052  1]                        Flags : 01
[035h 0053  1]                     Reserved : 00
[036h 0054  2]           PCI Segment Number : 0000
```

```
[038h 0056  8]        Register Base Address : 00000000FBFFE000
[040h 0064  1]      Device Scope Entry Type : 03
[041h 0065  1]                Entry Length : 08
[042h 0066  2]                    Reserved : 0000
[044h 0068  1]              Enumeration ID : 06
[045h 0069  1]              PCI Bus Number : F0
[046h 0070  2]                    PCI Path : [1F, 07]
[048h 0072  1]      Device Scope Entry Type : 03
[049h 0073  1]                Entry Length : 08
[04Ah 0074  2]                    Reserved : 0000
[04Ch 0076  1]              Enumeration ID : 07
[04Dh 0077  1]              PCI Bus Number : 00
[04Eh 0078  2]                    PCI Path : [13, 00]
[050h 0080  2]               Subtable Type : 0001 <Reserved Memory Region>
[052h 0082  2]                      Length : 0058
[054h 0084  2]                    Reserved : 0000
[056h 0086  2]          PCI Segment Number : 0000
[058h 0088  8]                Base Address : 00000000000EC000
[060h 0096  8]          End Address (limit) : 00000000000EFFFF
```

～ 略 ～

Hardware Unit Definition と表示されているのが DRHD で、Reserved Memory Region と表示されているのが RMRR です。

このテーブルの情報が誤っていると、BIOS とカーネルで VT-d を有効にしても Linux カーネルがエラーを起こして PCI パススルーが正常に動作しない場合があります[*4]。

VT-d のレジスタ

VT-d で使用される主なレジスタを表 3 に示します。VT-d のレジスタはメモリマップドでアクセスでき、ベースアドレスは前述の DMAR 上の DRHD で通知されます。

offset	name	description
008h	capability register	VT-d 上の機能の有無を示す
010h	extended capability register	追加の capability レジスタ
018h	global command register	VT-d の操作を行う
01ch	global status register	VT-d のステートを通知

[*4] この場合、ユーザの設定ミスではなく BIOS のバグなので、対策としてはサーバベンダから BIOS アップデートを受け取るか、カーネル側で DMAR を無視して強引に初期化するような方法しかありません。

offset	name	description
020h	root table address register	Root table のアドレスを設定
028h	context command register	context-cache を操作
034h	fault status register	フォールトを通知
XXXh	invalidate address register	IOTLB invalidate 時のアドレス指定
		invalidate address register のアドレスは
		extended capability register で定義
XXXh+008h	IOTLB invalidate register	IOTLB invalidate を実行

表 3: VT-d の主なレジスタ

VT-d の有効化

VT-d を有効化し、DMA リマップを行うには以下のような手順で設定を行います。

設定完了までウエイトするために、global status register の translation enable status にビットが立つまでループします。

1. メモリ上にルートテーブル、コンテキストテーブルを作成
2. root table address register に root table のアドレスを設定し、global command register に set root table pointer をセット（表４）して root table のアドレスを設定します。 設定完了までウエイトするために、global status register の root table pointer status にビットが立つまで（表５）ループします
3. IOTLB、Context-cache を invalidate します（細かい手順は省略します）
4. global command register に translation enable をセットして DMA リマッピングを有効化します

bits	field	description
31	translation enable	DMA remapping 有効・無効化
30	set root table pointer	root table pointer のセット・アップデート
29	set fault log	fault log pointer のセット・アップデート
28	enable advanced fault logging	advanced fault logging 有効・無効化
27	write buffer flush	write buffer を flush
26	queued invalidation enable	queue invalidation 有効・無効化
25	interrupt remapping enable	interrupt remaping 有効・無効化

bits	field	description
24	set interrupt remap table pointer	interrupt remap table pointer のセット・アップデート
23	compatibility format interrupt	compatibility format interrupt 有効・無効化
22:00	reserved	予約フィールド

表 4: global command register

bits	field	description
31	translation enable status	DMA remapping の状態
30	root table pointer status	root table pointer の状態
29	fault log status	fault log pointer の状態
28	advanced fault logging status	advanced fault logging の状態
27	write buffer flush status	write buffer flush の状態
26	queued invalidation enable status	queue invalidation enable の状態
25	interrupt remapping enable status	interrupt remaping の状態
24	interrupt remap table pointer status	interrupt remap table pointer の状態
23	compatibility format interrupt status	compatibility format interrupt の状態
22:00	reserved	予約フィールド

表 5: global status register

まとめ

今回は、VT-d の詳細について解説しました。次回からは、よりソフトウェア寄りの視点から仮想化を解説していきたいと思います。

ライセンス