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１５回 PCI パススルーその１「PCI パススルーと IOMMU」

はじめに

前回・前々回は、総集編として 2 回にわたり仮想化システムの全体像をふりかえりました。

今回は、PCI パススルーについて解説していきます。

PCI のおさらい

PCI パススルーの解説を行う前に、まずは簡単に PCI についておさらいしましょう。

PCI デバイスが持つ ID

PCI デバイスは Bus Number・Device Number で一意に識別され、1 つのデバイスが複数の機能を有する場合は Function Number で個々の機能が一意に識別されます。

Linux で “lspci -nn” を実行したときに出力の左端に表示される「aa:bb:c」のうち、aa が Bus Number、bb が Device Number、c が Function Number にあたります（リスト 1)。

さらに、そのデバイスがどこのメーカーのどの機種であるかという情報は Vendor ID・Device ID で表され、この情報によって OS はロードするドライバを選びます。Linux で lspci -nn を実行したときに出力の右端に表示される「dddd:eeee」のうち、dddd が Vendor ID、eeee が Device ID にあたります。

リスト 1, lspci **実行例**

```
$ lspci  -nn|grep IDE
00:1f.2 IDE interface [0101]: Intel Corporation 82801JI (ICH10 Family) 4 port SA
TA IDE Controller #1 [8086:3a20]
00:1f.5 IDE interface [0101]: Intel Corporation 82801JI (ICH10 Family) 2 port SA
TA IDE Controller #2 [8086:3a26]
```

PCI デバイスが持つメモリ空間

これらのデバイスは PCI Configuration Space、PCI I/O Space、PCI Memory Space の 3 つのメモリ空間を持ちます。 PCI Configuration Space はデバイスがどこのメーカーのどの機種であるかを示す Vendor ID・Device ID や、PCI I/O Space・PCI Memory Space のマップ先アドレスを示す Base Address Register、MSI 割り込みの設定情報など、デバイスの初期化とドライバのロードに必要な情報を多数含んでいます。

PCI Configuration Space にアクセスするには、次のような手順を実施する必要があります。

1. デバイスの Bus Number・Device Number・Function Number とアクセスしたい領域のオフセット値を EnableBit とともに CONFIG_ADDRESS レジスタ[*1]にセットする（CONFIG_ADDRESS レジスタのビット配置は表 [tab1] のとおり）
2. CONFIG_DATA レジスタ[*2]に対して読み込みまたは書き込みを行う

OS は PCI デバイス初期化時に、Bus Number・Device Number をイテレートして順に PCI Configuration Space の Vendor ID・Device ID を参照することで、コンピュータに接続されている PCI デバイスを検出できます。

PCI I/O Space は I/O 空間にマップされており（図 2)、おもにデバイスのハードウェアレジスタをマップするなどの用途に使われているようです[*3]。

PCI Memory Space は物理アドレス空間にマップされており、ビデオメモリなど大きなメモリ領域を必要とする用途に使われているようです[*4]。

どちらの領域もマップ先は PCI Configuration Space の Base Address Register を参照して取得する必要があります。

bit	name
31	Enable Bit
30-24	Reserved
23-16	Bus Number
15-11	Device Number
10-8	Function Number
7-2	Register offset

[*1] PC では CONFIG_ADDRESS レジスタは I/O 空間の 0xCF8 にマップされています。
[*2] PC では CONFIG_DATA レジスタは I/O 空間の 0xCFC にマップされています。
[*3] PC では I/O 空間にマップされますが、ほかのアーキテクチャではメモリマップされる場合もあると思われます。
[*4] 必ずしもこのように使い分けられているわけではなく、ハードウェアレジスタのマップに PCI Memory Space を使用するデバイスも存在します。

bit	name

表 1: CONFIG_ADDRESS register

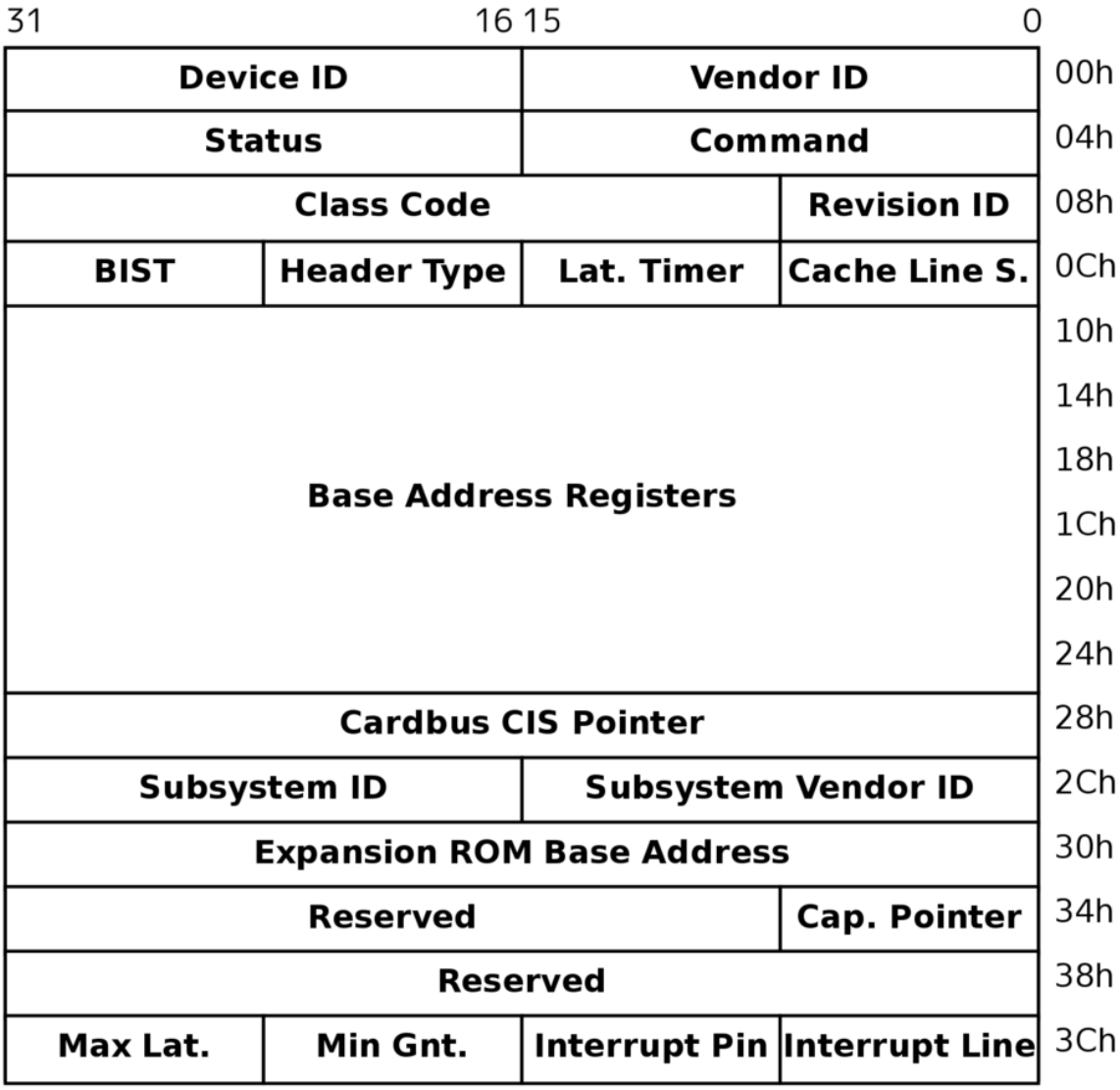

図 1　PCI Configuration Space

PCI デバイスにおける DMA 機能

PCI デバイスが持つメモリ空間に対して読み書きを行うことで、OS 上のデバイスドライバとデバイスの間でデータをやりとりできます。具体的には、I/O 空間に対してなら in/out 命令を発行、物理アドレス空間ならアドレスに対して memcpy() のような処理を行うことでデータを転送します。しかしながら、この方法ではブ

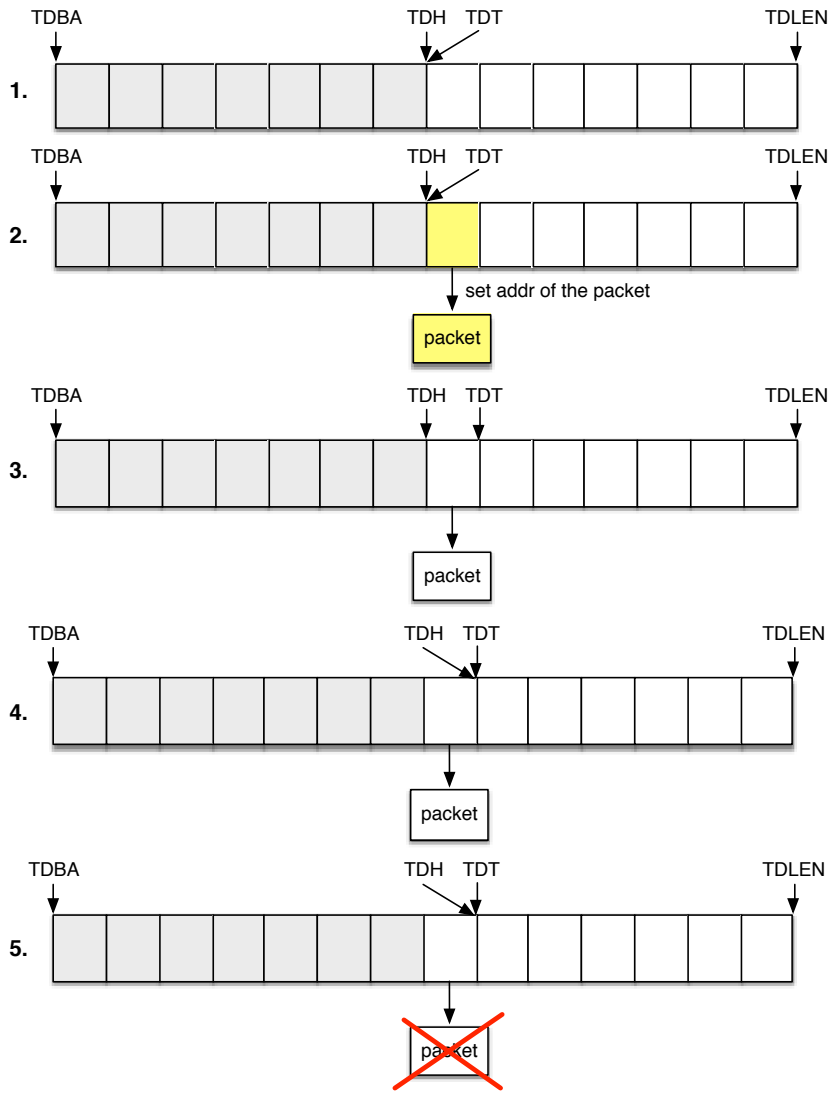

図 2　82583V GbE Controller のパケット送信機能

ロックデバイスや高速な NIC など、短い時間に大量のデータを転送する用途ではデータ転送を行う度に CPU が占有されてしまい、十分な性能が得られません。このような PCI デバイスでは DMA 機能が使用されます。

例として、82583V GbE Controller(e1000e) のパケット送信機能における DMA の使われ方を見てみましょう。図 3 に 82583V の送信用リングバッファと 4 種類のハードウェアレジスタ（TDBA、TDLEN、TDH、TDT）の関係を示しました。

TDBA はリングバッファの先頭アドレスを指します。TDLEN はリングバッファ長を示します。TDH は NIC がリングバッファのどこまで送信完了したかを示します。TDT はドライバがリングバッファのどこまでパケットを積んだかを示します。

次にパケット送信の手順を示します。手順で用いる番号は図 3 の番号に対応しています。

1. パケット送信処理を行う直前のレジスタおよびリングバッファの状態
2. ドライバはパケットを受け取り、TDT の次のエントリにパケットバッファのアドレスを書き込む
3. ドライバは TDT を 1 つ進めて、NIC へ送信可能なパケットがあることを伝える
4. NIC は TDT への書き込みを受けて、パケットをメインメモリから NIC 上のバッファへ DMA 転送し、送出する。送信が終わったら TDH を 1 つ進めて送信完了割り込みを起す
5. ドライバは送信完了割り込みを受け、送信済みパケットバッファ（TDH より手前にあるパケットすべて）に割り当てられたメモリを解放する

上述の例では話をわかりやすくするため、登場するパケットは 1 つだけにしましたが、実際には NIC の送信状況にかかわらずドライバはどんどんリングバッファへパケットを積んでいきます[*5]。NIC は順にパケットを DMA 転送して、送信が完了したら送信完了割り込みを行います。ドライバが TDT へ書き込んでから NIC が送信を完了するまでの間、OS は送信完了を待つ必要はありません。この間、OS はほかの処理に CPU の時間を使えます。送信に使ったパケットバッファを片付けるために、OS は NIC からの送信完了割り込みを使います。

PCI デバイスのパススルー

PCI Configuration Space のパススルー

ゲスト OS による CONFIG_ADDRESS レジスタと CONFIG_DATA レジスタへのアクセスを VMExit Reason 30（I/O Instruction）の VMExit を用いて VMM でハンドルします[*6]。

VMM は CONFIG_ADDRESS レジスタに設定された Bus Number・Device Number・Function Number からどのデバイスへアクセスが来たのかを識別し、パススルー対象デバイス宛てであれば、実デバイスの Configuration Space の指定オフセットへアクセスを行います。書き込みであれば CONFIG_DATA レジスタの値を実デバイスへ書き込み、読み込みであれば実デバイスから読み込んで CONFIG_DATA レジスタから返します。

PCI I/O Space のパススルー

ゲスト OS によるパススルーデバイスの PCI I/O Space へのアクセスも、同じく VMExit Reason 30（I/O Instruction）の VMExit を用いて VMM でハンドルします。このとき対象デバイスが持っている I/O ポートの範囲は PCI Configuration Space の Base Address Register で定義されており、VMM はこの値を記憶しておくことでどのデバイスへの I/O 命令であったか判別できます。

VMM はゲストからデバイスのポートへ I/O 命令を受け、同じポートへ I/O 命令を発行し結果をゲストに返します。あるいは、ゲスト側とホスト側で I/O ポートの番号が一致している場合[*7]は、前述の方法をとらずに

[*5] ただし、リングバッファが一杯になってしまったら一時的に送信を抑制するなどの措置をとります。
[*6] 詳しくは第 3 回 I/O 仮想化「デバイス I/O 編」を参照。
[*7] VMM が独自の Base Address Register を用意しておらず、実デバイスのものをそのままゲストに見せている場合。

VMCS の VM-Execution Control Fields にある I/O-Bitmap で対象となるポート番号のビットを 0 に設定することで VMExit を起こさずに PCI デバイスへアクセスできます。

PCI Memory Space のパススルー

ゲスト OS による PCI Memory Space へのアクセスは、ゲストマシンの物理メモリ空間のページ割り当てを設定することで実現します。本連載の『第 3 回 I/O 仮想化「デバイス I/O 編」』にて、EPT によるメモリマップド I/O のハンドリング方法として、デバイスがマップされたアドレスへのアクセスが発生したときに VMExit reason 48（EPT violation で、VMExit させる方法）を示しました[*8]。

このように無効なページを作って VMExit させるのではなく、デバイスがマップされたページを有効なページとして設定[*9]し、実デバイスのメモリマップされたエリアをマップします。これにより、ゲスト OS は VMExit されることなく直接実デバイスの PCI Memory Space に対してアクセスできます。

割り込みのパススルー

実 PCI デバイスからの実割り込みは VMExit reason 1（External Interrupt）を生じさせ、VMM によって[*10]ハンドルされます。VMM はこの割り込みを処理するためのパススルー専用割り込みハンドラをあらかじめ登録しておく必要があります。割り込みハンドラはこの割り込みを受け付け、ゲスト OS へ割り込みが届いたことを伝えるために、ゲストマシンの Local APIC レジスタを更新します。 そして VMCS の VM-entry interruption-information field へ割り込みをセットして VMEntry します[*11]。これにより、VMM で受け取った割り込みがゲストへ伝えられ、ゲスト OS が割り込みハンドラを実行できます。

PCI パススルーで DMA 転送時に生じる問題

ここまで、PCI メモリ空間や割り込みは一定の手順を踏めばパススルーできるということを示してきましたが、DMA では 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。通常、物理メモリ領域の全域にアクセスできます。もちろん、PCI デバイスはデバイスを使用している OS が仮想化されていることなど知りません。

この状態で、図 4 のゲスト A 上のドライバが DMA 先アドレスとしてゲスト物理ページの 2 番を指定すると何が起こるでしょうか？ PCI デバイスは DMA リクエスト元の OS が仮想化されていることを知らないので、ホスト物理ページの 2 番へ DMA 転送を行います。結果、PCI デバイスはゲスト A のメモリ領域の範囲外にデータを書き込んでしまいます。

[*8] シャドーページングのときも同じ要領で設定ができますが、今回は解説を割愛します。
[*9] Read access ビット・Write access ビットをともに 1 にします。
[*10] KVM・BHyVe のようにホスト OS 上で VMM が動く場合はホストカーネルによって違います。
[*11] 詳しい仮想割り込みのセット方法は第 5 回 I/O 仮想化「割り込み編・その 2」を参照。

これにより、別のゲストマシンや VMM のメモリ領域を破壊してしまいます。かといって、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られません。たとえ持っていたとしても、ゲスト OS が悪意を持ってゲストマシンに割り当てられている範囲外のページ番号を指定することで他のゲストマシンや VMM のメモリ領域を破壊できるという問題が解決しません。

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、図 4 の例ではゲスト A のメモリ領域の範囲外へデータを書き込んでしまっていた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 5)。

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。VT-d 対応 PC では、VMM が IOMMU へ変換テーブルを設定してアドレス変換を行い、ゲストマシンへの安全なデバイスパススルーを実現できます。

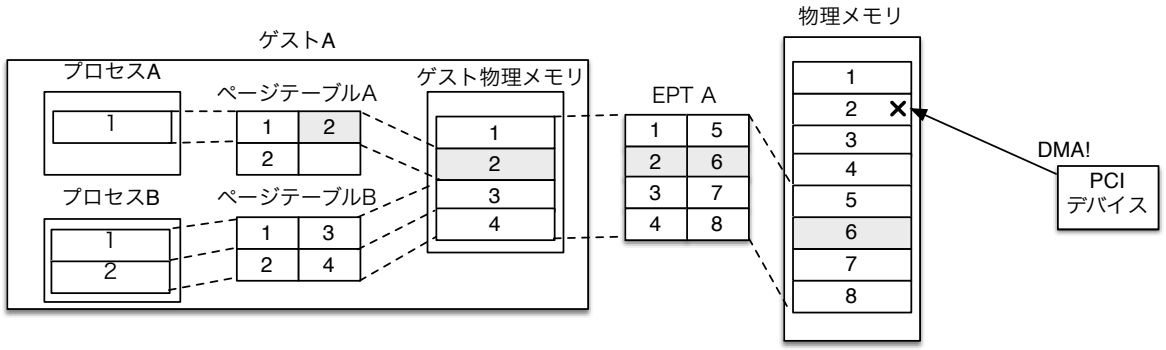

図 3　PCI パススルーで DMA 転送時に生じる問題

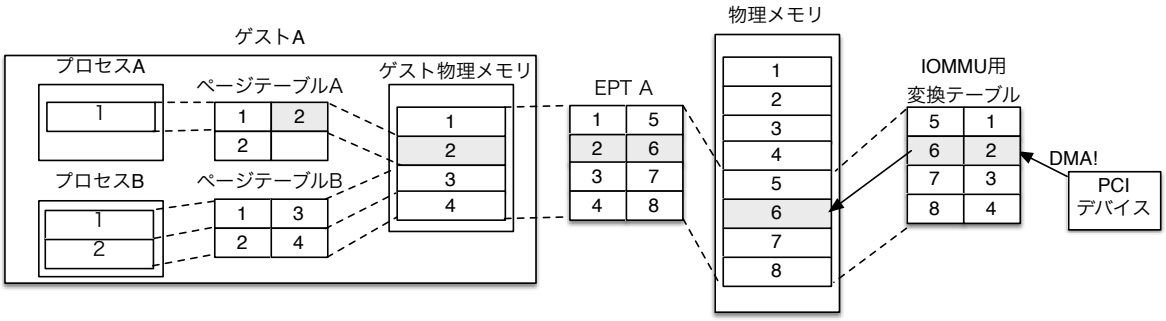

図 4　IOMMU を用いた DMA 時のアドレス変換

まとめ

今回は、PCI のおさらいと PCI パススルーの実現方法、IOMMU について解説しました。次回は、Intel VT-d の詳細について解説します。

ライセンス