Xen Multi-Function PCI Pass-Through

Simon Horman <simon@valinux.co.jp>

VA Linux Systems Japan K.K.

v1.0: 17th June 2009
v1.1: 6th October 2009

1 Introduction

PCI Pass-Through is a method of making a PCI function available to a domain. It works by assigning the resources of the function to exclusively to the domain. This is as opposed to emulation, whereby a virtual device is provided to arbitrate access to a physical device.

Pass-through is typically done with the help of an IOMMU which isolates the resources of the function in question, less the otherwise unprivileged domain escalate its privileges through the DMA capabilities that come with access to a PCI function.

The work discussed here focused on fully-virtualised domains however pass-through may also be used with paravirtualised domains.

1.1 Multi-Function and Single-Function

For the purposes of this discussion it is useful to classify PCI devices into two groups, single-function devices which as the name suggests only contain one PCI function; and multi-function devices which have between two and eight functions.

PCI functions are numbered from zero through to seven. All PCI devices must have function-zero present. So a single-function device's function will always be function-zero. And a multi-function device will always have function-zero and at least one of functions two to seven. A multi-function device's function numbers do not need to be continuous. For instance a device with functions zero, one and seven is valid.

1.2 Multi-Function in Xen Guests

Xen currently allows physical PCI functions to be passed-through as single-function devices. That is, regardless of the physical function number a function will appear as function-zero of a single-function device in the unprivileged domain that it has been passed-through to.

The aim of this work is to allow a group of passed-through PCI functions which belong to the same multi-function physical device to appear as a multi-function device when passed-through to an unprivileged domain. This will be referred to as multi-function pass-through for the remainder of this discussion.

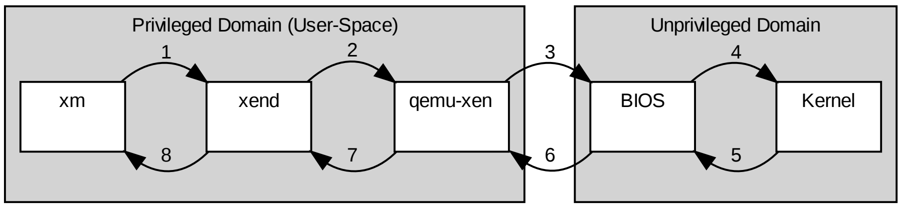

Figure 1: Event Flow During PCI Attach and Detach

2 Pass-Through Architecture

Xen provides four different operations that relate to PCI pass-through: attachment, detachment, listing of attached devices and listing of attachable devices.

Attachment of a pass-through device may occur when a domain is created. This is often referred to as *static assignment*, however this is misleading as the device may be subsequently unassigned. For this reason *domain-creation-time assignment* will be used in this discussion. Attachment may also occur after domain-creation. This is is usually referred to as hot-plug as it bears some similarity to the concept of physically hot-plugging PCI add-on cards.

Detachment of a pass-through device may occur after a domain has been created. It is often referred to as hot-unplug and is logically the opposite of hot-plug. However the logic concepts are not completely symmetrical as any function that has been passed-through may be detached, regardless of if it was attached at domain-creation-time or subsequently using hot-plug.

Up until recently domain-creation-time assignment and hot-plug used different code-paths within xend and different methods of communicating with qemu-xen. This meant any changes to assignment code generally had to be made twice. In order to simplify both the existing code and future changes domain-creation-time assignment has been changed to use the hot-plug code.

None the less, attachment and detachment are complex operations that ultimately trigger an event in the unprivileged domain's kernel. As illustrated in figure 1 the steps by by sub-system are as follows:

1. xm is a command-line tool. It accepts commands from the end-user and makes corresponding requests to xend.
2. xend is a management daemon that marshals information between the different sub-systems of a xen system. In this case it checks the pass-through commands sent by xm, reconciles them with the current state of the system, and passes them on to qemu-xend.
3. qemu-xen is used to emulate devices and control pass-through devices. Here it accepts pass-through attach or detach commands from qemu-xend. It reconfigures the xen hypervisor accordingly and triggers a corresponding ACPI event in the virtual BIOS of the target unprivileged domain.
4. The BIOS in turn triggers an ACPI event in the kernel of the unprivileged domain.
5. The unprivileged domain's kernel receives the ACPI event, hot-plugs or unplugs the device, and sends an acknowledgement back to the BIOS.
6. The BIOS passes on an acknowledgement to qemu-xen.

7. qemu-xen updates its internal state and passes on an acknowledgement to xend.
8. xend updates its internal state and that of xenstore, and passes on an acknowledgement to xm.

Listing of attached devices is a simpler operation that is effected by xend returning the relevant portions of the system state to xm. Listing of attachable devices is simpler still, xm inspects sysfs.

3 Implementation Challenges

A number of challenges presented themselves while implementing multi-function pass through. While individually quite simple, their interaction made things more difficult than was originally expected.

3.1 User interaction

The notation of specifying devices to be attached and detached needs to be extended in order to allow for multi-function devices. The approach taken has been to come up with a solution that limits the scope for user-space error while exposing the full scope of multi-function pass-through capability. With this in mind BDF notation has been extended to allow a multi-function device to be described as a single atomic unit.

3.1.1 Simple BDF Notation

BDF stands for Bus Device Function and is a notation used to succinctly describe PCI and PCIe devices. The simple form of the notation is:

- PCI Bus number in hexadecimal, often padded using a leading zeros to two or four digits
- A colon (:)
- PCI Device number in hexadecimal, often padded using a leading zero to two digits . Sometimes this also referred to as the slot number.
- A decimal point (.)
- PCI Function number in hexadecimal.

For example, 00:02.0 describes bus 0, device 2, function 0.

3.1.2 Extended BDF Notation

There is a common extension to this which includes the PCI domain number. The extended syntax is formed by optionally prefixing the notation above with:

- PCI domain number, often padded using leading zeros to four digits

- A colon (:)

For example, 0000:00:02.0 describes domain 0, bus 0, device 2, function 0.

Confusingly, PCI domains do not correspond to Xen domains. PCI domains are a physical property of the host, as are PCI buses.

3.1.3 Extended BDF Notation with Virtual-Slots

Xen further extends BDF notation to allow a virtual-slot to be supplied. This notation is used when specifying domain-creation-time attachment with the configuration file of a domain. A side-effect of the work on multi-function is that this notation is also now accepted for hot-plug as well.

This notation is formed by optionally appending he following to extended BDF:

- An at-sign (@)
- A virtual slot number in hexadeximal, may be padded using leading zeros to two digits

Or any number of:

- A comma (,)
- An option name. Currently the only valid option names are msitranslate and power_mgmt.
- An equals-sign (=)
- A value for the option. Currently the only valid values are 0 or 1, and yes or no.

In the case where both a virtual-slot and options are specified, the virtual-slot must come first.

For example, 0000:00:02.0@1c,msitranslate=1 denotes that physical function 0000:00:02.0 should be passed-through into virtual-slot 1c and that the resulting pass-through device will have the msitranslate option enabled..

3.1.4 Extension of BDF Notation for Multi-Function

The proposed extension to BDF format is to expand the function field to accept a comma-delimited list of:

- Function numbers
- A range of function numbers, denoted by:
  - The first function number
  - A hyphen (-)
  - The last function number

- An asterisk (*), used to denote all physical functions that are present in the device

None of the characters used in this expanded syntax for a function are valid in option names, so parsing can be done unambiguously.

This notation is internally expanded into groups of functions. For example:

Multi-Function Notation	Physical	Virtual
0000:00:1d.0-2	0000:00:1d.0	0000:00:7.0
	0000:00:1d.1	0000:00:7.1
	0000:00:1d.2	0000:00:7.2
0000:00:1d.0,3,5,7	0000:00:1d.0	0000:00:7.0
	0000:00:1d.3	0000:00:7.3
	0000:00:1d.5	0000:00:7.5
	0000:00:1d.7	0000:00:7.7
0000:00:1d.*	0000:00:1d.0	0000:00:7.0
	0000:00:1d.1	0000:00:7.1
	0000:00:1d.2	0000:00:7.2
	0000:00:1d.3	0000:00:7.3
	0000:00:1d.5	0000:00:7.5
	0000:00:1d.7	0000:00:7.7

These examples assume that virtual slot 7 will be used. The last example is for a physical device that has functions 0, 1, 2, 3, 5 and 7.

3.1.5 Extension of BDF Notation for Multi-Function with Explicit Vfunctions

In order to allow control over the mapping of physical to virtual functions a further extension to BDF notation is provided.

In this notation physical function numbers are replaced by function units which comprise of:

- Physical function number and optionally;
- An equals sign and;
- A virtual function number to use

For example:

Multi-Function Notation	Physical	Virtual
0000:00:1d.2=0-0=2	0000:00:1d.2	0000:00:7.0
	0000:00:1d.1	0000:00:7.1
	0000:00:1d.0	0000:00:7.2
0000:00:1d.0=3,3=2,5=1,7=0	0000:00:1d.7	0000:00:7.0
	0000:00:1d.5	0000:00:7.3
	0000:00:1d.3	0000:00:7.5
	0000:00:1d.0	0000:00:7.7

Again examples assume that virtual slot 7 will be used. And the last example is for a physical device that has functions 0, 1, 2, 3, 5 and 7.

When specifying virtual functions it should be noted that:

- A virtual device must include virtual function zero.
- It may be possible to construct a virtual device whose functions do not operate correctly due to hardware limitations.

A limitation of this scheme is that it does not allow functions from different physical devices to be combined to form a multi-function virtual device. However it is of the concern that such combinations would be invalid due to hardware limitations. This needs further investigation.

3.2 Mapping Physical-Functions to Virtual-Functions

At this time the mapping of physical to virtual function numbers is done by xen. At this time two mapping schemes have been proposed.

3.2.1 Identity Mapping

Physical function numbers are identity mapped into virtual functions.

For example the following physical to virtual mapping may occur. Here the treatment of the function numbers is critical and will always be consistent.

Physical	Virtual
00:1d.0	00:07.0
00:1d.1	00:07.1
00:1d.7	00:07.7

As function-zero must be present, this scheme only allows one multi-function pass-through device per physical device.

3.2.2 Least Mapping

The lowest available virtual function is used.

Physical	Virtual
00:1d.0	00:07.0
00:1d.1	00:07.1
00:1d.7	00:07.2

Unlike identity mapping, this scheme is dependant on the order that functions are supplied. So if the order in the above example was reversed, then the result would be:

Physical	Virtual
00:1d.7	00:07.0
00:1d.1	00:07.1
00:1d.0	00:07.2

As physical function-zero need not be present, this scheme allows multiple virtual multi-function devices to be formed from functions of the same physical multi-function device.

It is not known if this scheme allows the construction of mutli-function devices that violate hardware restrictions. It seems that this is more likely under the least-mapping scheme than the more restrictive identity-mapping scheme.

The least mapping scheme was proposed by Jun Kamada of Fujitsu.

3.2.3 Identity Mapping with Override

This mapping scheme aims to combine the simplicity of identity mapping with the flexibility of least mapping. It came about when the ability to denote explicit virtual-functions was added to the BDF notation for multi-function devices. It works in three phases:

- Use any virtual functions denoted in the BDF notation
- Then, map the lowest remaining physical function to virtual function 0 as needed
- Finally, identity map the rest of the functions

For example:

Multi-Function Notation	Physical	Virtual
0000:00:1d.1,3,4,5=7	0000:00:1d.1	0000:00:7.0
	0000:00:1d.3	0000:00:7.3
	0000:00:1d.4	0000:00:7.4
	0000:00:1d.5	0000:00:7.7

It fails if it can't maintain the property that:

- A virtual device must always include virtual function zero

Identity mapping with override is the scheme that is currently merged into Xen.

3.3 Passing Virtual-Functions Around

An interesting problem that arose while implementing multi-function pass-through is that of passing around virtual function information.

3.3.1 Encoding Virtual-Functions

The first component of this problem is how to encode the information as the existing xen pass-through code deals with virtual-slots.

Inside the Linux kernel the concept of a devfn — the combination of a device (slot) and function number — is used. This is convenient as it maps to the lowest 8 bits of a functions PCI identifier, the top five bits being the device and the lower 3 bits being the function. It seemed logical to take the xen code and simply use a virtual-devfn in each place where a virtual-slot was used. Although quite a noisy change in terms of renaming variables, it was reasonably straight-forward to ensure that all the relevant code was updated.

3.3.2 Allocating Virtual-Functions

A second part of the problem of passing virtual-function information around is how virtual-functions are allocated.

Virtual-Slots are allocated inside qemu-xen as it knows which slots are free. Initially it seemed logical to also perform virtual-function allocation within xend. However this was complicated by the decision to have xend marshal the component functions of a multi-function pass-through device. Instead virtual-function numbers are assigned by xm at the time that BDFs for pass-through devices are parsed. This allows for simple handling of cases where virtual-functions are supplied in the BDF.

Auto-allocation of virtual-functions occurs at the same point regardless of if the virtual-slot will be automatically assigned or not and the request is always encoded as a extended devfn. This is achieved by using bit nine of the devfn as a flag. If the virtual-slot is to be automatically allocated by qemu-xen, then the flag is set to one and the virtual-slot bits of the devfn are ignored. Otherwise the flag is set to one and the virtual-slot bits of the devfn are set to the user-supplied value. As bit nine is outside the range of an normal devfn, which is 8 bits long in denotes that auto allocation should occur without using any potentially valid devfn values.

	flag (bit 9)	device/slot (bits 3–7)	function (bits 0–2)
user-supplied virtual-slot	0	user-supplied value	generated or user-supplied value
automatic virtual-slot assignment	1	0	generated or user-supplied value

Figure 2: Extended Devfn

Unfortunately this is not enough for qemu-xen to correlate the different component functions of a multi-function devices. Again this marshalling is done by xm/xend, though the procedure is far from clean.

1. Find all the functions with the same key, these will be all the functions of a pass-through device regardless of it is single or multi-function.
2. Order the functions so that virtual-function zero is last.
3. Insert the first function
4. If there are no more functions, finish — it is a single-function device.
5. Else, if the virtual-slot is to be automatically assigned, request the virtual function of the function that was just inserted and use this to set the virtual-slot of all remaining functions.
6. Insert each of the remaining functions.

3.4 ACPI restrictions

When performing PCI hot-plug and hot-unplug using ACPI, which is the foundation of how qemu-xen signals attachment and detachment of pass-through devices to the unprivileged domain via the virtual BOIS, each function of a multi-function device is dealt with as a separate ACPI event. The ordering of these events appears to be unimportant so long as the event for function-zero comes last.

While conceptually quite simple, it is worth noticing that the pass-through code in the BIOS, qemu-xen and a lot of xend deals with PCI functions not devices. In the case of a single-function device there is a one-to-one mapping between functions and devices. However in the case of multi-function devices higher level control is required at some point in order to group together functions belonging to a multi-function device and ensure that when attachment and detachment is preformed the event for function-zero occurs last.

The current design implements this grouping by associating a unique key for each device. This allows the existing model of dealing with functions to remain, and when any grouping is required it can be done by finding the devices with a common key and then reordering the list of requests for those devices accordingly.

4 Conclusion

The current implementation of multi-function pass-through is an incremental improvement, allowing physical multi-function devices to appear as multi-function devices in unprivileged guests. Moving forward it would be interesting to examine combining functions from different physical devices into a virtual multi-function device.