Question on MMIO for a PCI device

OSR_Community_User Member
Posts: **110,217**
November 2008

Hi,
I have a few questions about Memory mapped IO.
1. where does that intelligence of mapping physical address space to device
address lie?
2. What operations are involved (from a programmer's perspective)in
accomplishing this on a PCI device?
3. if I just modify the MMIO BAR of the PCI register, does it cause a
different block physical address space to be mapped to device IO?

regards,
Madhu

Tim_Roberts Member - All Emails
Posts: **13,871**
November 2008

Madhusudan Narayan wrote:
>
> I have a few questions about Memory mapped IO.
> 1. where does that intelligence of mapping physical address space to
> device address lie?

The responsibility is somewhat spread out. The device embeds in the BAR
the size of the region it needs. The BIOS then assigns a chunk of the
physical address space by writing to the BAR. Depending on the version,
the operating system might assign a new address by rewriting the BAR.
The driver has the job of mapping this physical address region into
virtual address space so that it can be used. Finally, it is the job of
the hardware to recognize and handle bus cycles that are within its BARs.

> 2. What operations are involved (from a programmer's perspective)in
> accomplishing this on a PCI device?

See previous paragraph. The samples show how to map the BAR into
virtual address space.

> 3. if I just modify the MMIO BAR of the PCI register, does it cause a
> different block physical address space to be mapped to device IO?

Only in the sense that the device will start looking for a different
region. The BIOS and the operating system will not know of the changes,
so they might think that the address space was usable for other
devices. Plus, if your device sits behind a PCI bridge, the bridge also
has to be notified of the new BAR. General advice: DO NOT change your
device's BAR. Let the operating system handle it.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

Tim Roberts, timr@probo.com
Providenza & Boekelheide, Inc.

OSR_Community_User Member
Posts: **110,217**
November 2008

Thanks a lot Tim.

Some clarification required(questions *inline*)
On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts wrote:

> Madhusudan Narayan wrote:
> >
> > I have a few questions about Memory mapped IO.
> > 1. where does that intelligence of mapping physical address space to
> > device address lie?
> The responsibility is somewhat spread out. The device embeds in the BAR
> the size of the region it needs. The BIOS then assigns a chunk of the
> physical address space by writing to the BAR. Depending on the version,
> the operating system might assign a new address by rewriting the BAR.
> The driver has the job of mapping this physical address region into
> virtual address space so that it can be used.

> Finally, it is the job of
> the hardware to recognize and handle bus cycles that are within its BARs.

*Here, hardware in the sense, the bus controller/Northbridge? Does this
remember what values were written into BARs?*

>
>
> > 2. What operations are involved (from a programmer's perspective)in
> > accomplishing this on a PCI device?
>
> See previous paragraph. The samples show how to map the BAR into
> virtual address space.
>
>
> > 3. if I just modify the MMIO BAR of the PCI register, does it cause a
> > different block physical address space to be mapped to device IO?

>
> Only in the sense that the device will start looking for a different
> region. The BIOS and the operating system will not know of the changes,
> so they might think that the address space was usable for other
> devices.

> Plus, if your device sits behind a PCI bridge, the bridge also
> has to be notified of the new BAR.

*How do we(rather OS/BIOS) notify the bridge? what is this bridge enumerated
as?*

> General advice: DO NOT change your
> device's BAR. Let the operating system handle it.
>
> --
> Tim Roberts, xxxxx@probo.com
> Providenza & Boekelheide, Inc.
>
>
> ---
> NTDEV is sponsored by OSR
>
> For our schedule of WDF, WDM, debugging and other seminars visit:
> http://www.osr.com/seminars
>
> To unsubscribe, visit the List Server section of OSR Online at
> http://www.osronline.com/page.cfm?name=ListServer
>

Tim_Roberts Member - All Emails
Posts: **13,871**
November 2008

Madhusudan Narayan wrote:
>
> On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts <mailto:xxxxx@probo.com>> wrote:
>
>
> Finally, it is the job of
> the hardware to recognize and handle bus cycles that are within
> its BARs.
>
>
> *Here, hardware in the sense, the bus controller/Northbridge?*

No, hardware in the sense of "your device". When the processor does a
read or write for an address that is not part of RAM, the request is
basically broadcast to the PCI bus by the bus controller. It is the
responsibility of each individual PCI device to monitor all those bus
cycles and "claim" the cycles that reside within the BARs they have been
assigned.

Things are a little different for PCI-Express, since that's a
point-to-point bus. In PCIe, the root complex routes the request to the
specific device instead of broadcasting it.

> *Does this remember what values were written into BARs?*

The device remembers this, since it's responsible for claiming the cycles.

>

>
> Plus, if your device sits behind a PCI bridge, the bridge also
> has to be notified of the new BAR.
>
>
> *How do we(rather OS/BIOS) notify the bridge? what is this bridge
> enumerated as?*

Generally, a PCI-to-PCI bridge will tell the bus controller which
regions it has reserved for devices behind the bridge. It's up to the
BIOS to make sure any devices behind that bridge live within the
reserved space. It's possible to change the bridge, but it is specific
to the chipset. The PCI bus driver knows how to do this, but the
functionality is not exposed to drivers, because it isn't needed. Only
the BIOS and the operating system should be assigning BARs.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

Tim Roberts, timr@probo.com
Providenza & Boekelheide, Inc.

f

OSR_Community_User Member
Posts: **110,217**
November 2008

Wow!!
Thanks a lot for clarifying.
Tim, Could you please provide some pointers where I can read more about it?

Thanks,
Madhu

On Tue, Nov 25, 2008 at 11:27 PM, Tim Roberts wrote:

> Madhusudan Narayan wrote:
> >
> > On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts > > wrote:
> >
> >
> > Finally, it is the job of
> > the hardware to recognize and handle bus cycles that are within
> > its BARs.
> >
> >
> > *Here, hardware in the sense, the bus controller/Northbridge?*
>
> No, hardware in the sense of "your device". When the processor does a
> read or write for an address that is not part of RAM, the request is
> basically broadcast to the PCI bus by the bus controller. It is the
> responsibility of each individual PCI device to monitor all those bus
> cycles and "claim" the cycles that reside within the BARs they have been
> assigned.
>
> Things are a little different for PCI-Express, since that's a
> point-to-point bus. In PCIe, the root complex routes the request to the
> specific device instead of broadcasting it.
>
>
> > *Does this remember what values were written into BARs?*
>
> The device remembers this, since it's responsible for claiming the cycles.
>
>
> >
> >
> > Plus, if your device sits behind a PCI bridge, the bridge also

> > has to be notified of the new BAR.
> >
> >
> > *How do we(rather OS/BIOS) notify the bridge? what is this bridge
> > enumerated as?*
>
> Generally, a PCI-to-PCI bridge will tell the bus controller which
> regions it has reserved for devices behind the bridge. It's up to the
> BIOS to make sure any devices behind that bridge live within the
> reserved space. It's possible to change the bridge, but it is specific
> to the chipset. The PCI bus driver knows how to do this, but the
> functionality is not exposed to drivers, because it isn't needed. Only
> the BIOS and the operating system should be assigning BARs.
>
> --
> Tim Roberts, xxxxx@probo.com
> Providenza & Boekelheide, Inc.
>
>
> ---
> NTDEV is sponsored by OSR
>
> For our schedule of WDF, WDM, debugging and other seminars visit:
> http://www.osr.com/seminars
>
> To unsubscribe, visit the List Server section of OSR Online at
> http://www.osronline.com/page.cfm?name=ListServer
>

f

OSR_Community_User Member
Posts: **110,217**
November 2008

Generally, you need to know nothing about the BARs. The translated
resources will be delivered to you in the IRP_MJ_PNP:IRP_MN_START
notification. The BAR values are established by the PCI BIOS, possibly
modified by *magic* in the operating system, and are not in the province of
the driver writer to change (for example, you not only have to change the
BAR registers, but then you have to reprogram all the bridge chips between
the memory controller and the device, not what is called Best
Practice...more along the lines of Suicidal Tendencies).

In modern drivers, you don't need to know anything about the BARs, and
ignore their existence. You see the translated registers when you parse the
resources block from the IRP cited above. In an old NT 4 legacy driver, you
had to do PCI bus enumeration, but the BARs were treated as read-only, and
you had to call HalTranslateBusAddress yourself (this is no longer done, and
the call is no longer used, because it is handled for you by the PnP
mechanism)
joe

-----Original Message-----
From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Tim Roberts
Sent: Tuesday, November 25, 2008 12:15 PM
To: Windows System Software Devs Interest List
Subject: Re: [ntdev] Question on MMIO for a PCI device

Madhusudan Narayan wrote:
>
> I have a few questions about Memory mapped IO.
> 1. where does that intelligence of mapping physical address space to
> device address lie?

The responsibility is somewhat spread out. The device embeds in the BAR
the size of the region it needs. The BIOS then assigns a chunk of the
physical address space by writing to the BAR. Depending on the version,
the operating system might assign a new address by rewriting the BAR.

The driver has the job of mapping this physical address region into
virtual address space so that it can be used. Finally, it is the job of
the hardware to recognize and handle bus cycles that are within its BARs.

> 2. What operations are involved (from a programmer's perspective)in
> accomplishing this on a PCI device?

See previous paragraph. The samples show how to map the BAR into
virtual address space.

> 3. if I just modify the MMIO BAR of the PCI register, does it cause a
> different block physical address space to be mapped to device IO?

Only in the sense that the device will start looking for a different
region. The BIOS and the operating system will not know of the changes,
so they might think that the address space was usable for other
devices. Plus, if your device sits behind a PCI bridge, the bridge also
has to be notified of the new BAR. General advice: DO NOT change your
device's BAR. Let the operating system handle it.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

NTDEV is sponsored by OSR

For our schedule of WDF, WDM, debugging and other seminars visit:
http://www.osr.com/seminars

To unsubscribe, visit the List Server section of OSR Online at
http://www.osronline.com/page.cfm?name=ListServer

--
This message has been scanned for viruses and
dangerous content by MailScanner, and is
believed to be clean.

OSR_Community_User Member
Posts: **110,217**
November 2008

If you are trying to program PCI and do not have a copy of the Mindshare bok
"PCI System Architecture" (Tom Shanley and Don Anderson, Addison Wesley; I
have the Third Edition, up to PCI 2.1, published in 1995. They also have
PCIe and PCIx books), then stop right now and go get them.

The Hub Controller Interface (nee North Bridge) only knows a window of
address values. It drops the address onto the address lines of the PCI bus.
After that, it is up to the devices to respond. If a device does not
acknowledge the address within a specified time, an access fault is
triggered. The HCI does not "remember" any BAR values at all; it only
handles the idea that some block of addresses are rumored to be on the bus,
and it is up to the individual devices to confirm (or implicitly deny) those
rumors.

Because each bridge chip *does* have an address range (a base register and a
limit register, see p.419 of the above-cited work), if you were to change
the BAR value, you would have to program EACH bridge chip along the path to
have a different limit.

Why do you think this is even a concept that anyone should care about? The
whole point of the OS is to mask this nonsense so you never need to worry
about it. The mechanisms for handling this are built deep into the BIOS and
OS, and are of no concern to you; all you care about is what VIRTUAL address

you have to access the MMIO values on the card, and that's already handed to you! There is absolutely no reason you need to write the BAR registers or reprogram the bridges. You are approaching this problem from the wrong direction. Stop thinking grubby-and-irrelevant-low-level hardware details and think only of the OS interface presented to you.

Look at some PCI examples in the DDK, for example, search for CM_RESOURCE_LIST in the toaster.c example to see where you get the virtual address (in my copy, it is around line 1346). This would give you an address/length pair for one of the addresses mapped from the BAR.
joe

_____

From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Madhusudan Narayan
Sent: Tuesday, November 25, 2008 12:39 PM
To: Windows System Software Devs Interest List
Subject: Re: [ntdev] Question on MMIO for a PCI device

Thanks a lot Tim.

Some clarification required(questions inline)
On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts wrote:
Madhusudan Narayan wrote:
>
> I have a few questions about Memory mapped IO.
> 1. where does that intelligence of mapping physical address space to
> device address lie?
The responsibility is somewhat spread out. The device embeds in the BAR the size of the region it needs. The BIOS then assigns a chunk of the physical address space by writing to the BAR. Depending on the version, the operating system might assign a new address by rewriting the BAR. The driver has the job of mapping this physical address region into virtual address space so that it can be used.

Finally, it is the job of
the hardware to recognize and handle bus cycles that are within its BARs.

Here, hardware in the sense, the bus controller/Northbridge? Does this remember what values were written into BARs?

> 2. What operations are involved (from a programmer's perspective)in
> accomplishing this on a PCI device?
See previous paragraph. The samples show how to map the BAR into virtual address space.

> 3. if I just modify the MMIO BAR of the PCI register, does it cause a
> different block physical address space to be mapped to device IO?
Only in the sense that the device will start looking for a different region. The BIOS and the operating system will not know of the changes, so they might think that the address space was usable for other devices.

Plus, if your device sits behind a PCI bridge, the bridge also
has to be notified of the new BAR.

How do we(rather OS/BIOS) notify the bridge? what is this bridge enumerated as?

General advice: DO NOT change your
device's BAR. Let the operating system handle it.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

--- NTDEV is sponsored by OSR For our schedule of WDF, WDM, debugging and other seminars visit: http://www.osr.com/seminars To unsubscribe, visit the List Server section of OSR Online at
http://www.osronline.com/page.cfm?name=ListServer

OSR_Community_User Member
Posts: **110,217**
November 2008

Joe,
I agree that driver programmers do not need to worry about these.
I was(am) just curious to know the little magic done by the OS, and BIOS.

Thanks a lot for mentioning the sources of more information.

regards,
Madhu

On Wed, Nov 26, 2008 at 12:21 AM, Joseph M. Newcomer
wrote:

> If you are trying to program PCI and do not have a copy of the Mindshare
> bok "PCI System Architecture" (Tom Shanley and Don Anderson, Addison
> Wesley; I have the Third Edition, up to PCI 2.1, published in 1995. They
> also have PCIe and PCIx books), then stop right now and go get them.
>
>
>
> The Hub Controller Interface (nee North Bridge) only knows a window of
> address values. It drops the address onto the address lines of the PCI
> bus. After that, it is up to the devices to respond. If a device does
> not acknowledge the address within a specified time, an access fault is
> triggered. The HCI does not "remember" any BAR values at all; it only
> handles the idea that some block of addresses are rumored to be on the bus,
> and it is up to the individual devices to confirm (or implicitly deny) those
> rumors.
>
>
>
> Because each bridge chip **does** have an address range (a base register
> and a limit register, see p.419 of the above-cited work), if you were to
> change the BAR value, you would have to program EACH bridge chip along the
> path to have a different limit.
>
>
>
> Why do you think this is even a concept that anyone should care about? The
> whole point of the OS is to mask this nonsense so you never need to worry
> about it. The mechanisms for handling this are built deep into the BIOS
> and OS, and are of no concern to you; all you care about is what VIRTUAL
> address you have to access the MMIO values on the card, and that's already
> handed to you! There is absolutely no reason you need to write the BAR

> registers or reprogram the bridges. You are approaching this problem from
> the wrong direction. Stop thinking grubby-and-irrelevant-low-level
> hardware details and think only of the OS interface presented to you.
>
>
>
> Look at some PCI examples in the DDK, for example, search for
> CM_RESOURCE_LIST in the toaster.c example to see where you get the virtual
> address (in my copy, it is around line 1346). This would give you an
> address/length pair for one of the addresses mapped from the BAR.
>
> joe
>
>
> ------------------------------
>
> *From:* xxxxx@lists.osr.com [mailto:
> xxxxx@lists.osr.com] *On** Behalf Of *Madhusudan Narayan
> *Sent:* Tuesday, November 25, 2008 12:39 PM
> *To:* Windows System Software Devs Interest List
> *Subject:* Re: [ntdev] Question on MMIO for a PCI device
>
>
>
> Thanks a lot Tim.
>
>
>
> Some clarification required(questions *inline*)
>
> On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts wrote:
>
> Madhusudan Narayan wrote:
> >
> > I have a few questions about Memory mapped IO.
> > 1. where does that intelligence of mapping physical address space to
> > device address lie?
>
> The responsibility is somewhat spread out. The device embeds in the BAR
> the size of the region it needs. The BIOS then assigns a chunk of the
> physical address space by writing to the BAR. Depending on the version,
> the operating system might assign a new address by rewriting the BAR.
> The driver has the job of mapping this physical address region into
> virtual address space so that it can be used.
>
>
>
>
>
> Finally, it is the job of
> the hardware to recognize and handle bus cycles that are within its BARs.
>
>
>
> *Here, hardware in the sense, the bus controller/Northbridge? Does this
> remember what values were written into BARs?*
>
>
>
>
>
> > 2. What operations are involved (from a programmer's perspective)in
> > accomplishing this on a PCI device?
>
> See previous paragraph. The samples show how to map the BAR into
> virtual address space.
>
>
>
> > 3. if I just modify the MMIO BAR of the PCI register, does it cause a

> > different block physical address space to be mapped to device IO?
>
> Only in the sense that the device will start looking for a different
> region. The BIOS and the operating system will not know of the changes,
> so they might think that the address space was usable for other
> devices.
>
>
>
> Plus, if your device sits behind a PCI bridge, the bridge also
> has to be notified of the new BAR.
>
>
>
> *How do we(rather OS/BIOS) notify the bridge? what is this bridge
> enumerated as?*
>
>
>
> General advice: DO NOT change your
> device's BAR. Let the operating system handle it.
>
> --
> Tim Roberts, xxxxx@probo.com
> Providenza & Boekelheide, Inc.
>
>

f

OSR_Community_User Member
Posts: **110,217**
November 2008

Nothing wrong with intellectual curiosity, but you'd be surprised how many
times I find people asking me about this in class with the *intention* of
going out there and fiddling with them. I always ask, "You're a hardware
designer, aren't you?" and the answer is always "yes". And it is also the
case that they sometimes were working on bare machines without even a decent
PCI BIOS (in their past) and think that they actually *have* to do this
stuff. Some are even offended that they can't.

joe

_____

From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Madhusudan Narayan
Sent: Tuesday, November 25, 2008 2:00 PM
To: Windows System Software Devs Interest List
Subject: Re: [ntdev] Question on MMIO for a PCI device

Joe,
I agree that driver programmers do not need to worry about these.
I was(am) just curious to know the little magic done by the OS, and BIOS.

Thanks a lot for mentioning the sources of more information.

regards,
Madhu
On Wed, Nov 26, 2008 at 12:21 AM, Joseph M. Newcomer
wrote:
If you are trying to program PCI and do not have a copy of the Mindshare bok
"PCI System Architecture" (Tom Shanley and Don Anderson, Addison Wesley; I
have the Third Edition, up to PCI 2.1, published in 1995. They also have
PCIe and PCIx books), then stop right now and go get them.

The Hub Controller Interface (nee North Bridge) only knows a window of
address values. It drops the address onto the address lines of the PCI bus.
After that, it is up to the devices to respond. If a device does not
acknowledge the address within a specified time, an access fault is
triggered. The HCI does not "remember" any BAR values at all; it only
handles the idea that some block of addresses are rumored to be on the bus,
and it is up to the individual devices to confirm (or implicitly deny) those
rumors.

Because each bridge chip *does* have an address range (a base register and a
limit register, see p.419 of the above-cited work), if you were to change
the BAR value, you would have to program EACH bridge chip along the path to
have a different limit.

Why do you think this is even a concept that anyone should care about? The
whole point of the OS is to mask this nonsense so you never need to worry
about it. The mechanisms for handling this are built deep into the BIOS and
OS, and are of no concern to you; all you care about is what VIRTUAL address
you have to access the MMIO values on the card, and that's already handed to
you! There is absolutely no reason you need to write the BAR registers or
reprogram the bridges. You are approaching this problem from the wrong
direction. Stop thinking grubby-and-irrelevant-low-level hardware details
and think only of the OS interface presented to you.

Look at some PCI examples in the DDK, for example, search for
CM_RESOURCE_LIST in the toaster.c example to see where you get the virtual
address (in my copy, it is around line 1346). This would give you an
address/length pair for one of the addresses mapped from the BAR.
joe

_____

From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Madhusudan Narayan
Sent: Tuesday, November 25, 2008 12:39 PM

To: Windows System Software Devs Interest List
Subject: Re: [ntdev] Question on MMIO for a PCI device

Thanks a lot Tim.

Some clarification required(questions inline)

On Tue, Nov 25, 2008 at 10:45 PM, Tim Roberts wrote:
Madhusudan Narayan wrote:
>
> I have a few questions about Memory mapped IO.
> 1. where does that intelligence of mapping physical address space to
> device address lie?
The responsibility is somewhat spread out. The device embeds in the BAR
the size of the region it needs. The BIOS then assigns a chunk of the
physical address space by writing to the BAR. Depending on the version,
the operating system might assign a new address by rewriting the BAR.
The driver has the job of mapping this physical address region into
virtual address space so that it can be used.

Finally, it is the job of
the hardware to recognize and handle bus cycles that are within its BARs.

Here, hardware in the sense, the bus controller/Northbridge? Does this
remember what values were written into BARs?

> 2. What operations are involved (from a programmer's perspective)in
> accomplishing this on a PCI device?
See previous paragraph. The samples show how to map the BAR into
virtual address space.

> 3. if I just modify the MMIO BAR of the PCI register, does it cause a
> different block physical address space to be mapped to device IO?
Only in the sense that the device will start looking for a different
region. The BIOS and the operating system will not know of the changes,
so they might think that the address space was usable for other
devices.

Plus, if your device sits behind a PCI bridge, the bridge also
has to be notified of the new BAR.

How do we(rather OS/BIOS) notify the bridge? what is this bridge enumerated
as?

General advice: DO NOT change your
device's BAR. Let the operating system handle it.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

Tim_Roberts Member - All Emails
Posts: **13,871**
November 2008

Madhusudan Narayan wrote:
>
> I agree that driver programmers do not need to worry about these.
> I was(am) just curious to know the little magic done by the OS, and BIOS.

It's not really very magic. The BIOS (or OS) decides where it's going
to put PCI devices (usually at high physical addresses), then doles out
chunks of that space based upon the device's needs, just like any other
memory manager. Setting the BAR registers is a trivial operation
involving I/O ports defined in the PCI specifications (CF8/CFC).

As Joe said, a driver writer should treat all of this as a black box.
Your driver will be told what physical address was assigned to the
board. You map it into virtual memory, then use it.

--
Tim Roberts, xxxxx@probo.com
Providenza & Boekelheide, Inc.

Tim Roberts, timr@probo.com
Providenza & Boekelheide, Inc.

Maxim_S._Shatskih Member
Posts: **10,396**
November 2008

>1. where does that intelligence of mapping physical address space to device address lie?

In the BAR address decoding hardware of the device.

>2. What operations are involved (from a programmer's perspective)in accomplishing this on a PCI
>device?

PCI.SYS reads the "resource requirements" from the config space and reports them to PnP.
PnP arbitrates them - i.e. assigns unique values not clashing with anything.
PCI.SYS takes the "resources" values from PnP and programs the BARs.

>3. if I just modify the MMIO BAR of the PCI register, does it cause a different block physical address
>space to be mapped to device IO?

Yes.

--
Maxim S. Shatskih
Windows DDK MVP
xxxxx@storagecraft.com
http://www.storagecraft.com

f

OSR_Community_User Member
Posts: **110,217**
November 2008

With respect to point 1, a device *detects* that a physical address on the bus is destined for it by using the hardware as described, but that's a tiny piece of the problem. Generating that physical address on the bus is the hard part. HalTranslateBusAddress would be responsible for creating the mapping from virtual address to physical address, by obtaining a block (perhaps only one page, perhaps many pages) of virtual address space and updating the memory maps to convert those virtual addresses to the physical addresses for the bus. Since the physical address is of little interest to the programmer, it is the job of the OS to hide the magic that makes all that happen.

With respect to point 3, setting a different BAR value will cause a different physical address space to be used, but also would require reprogramming all the bridge chips in between. What would cause this to happen, since my act of writing the BAR value would not in any way actually notify the OS that this has happened, nor the entire PCI subsystem that something has changed. So the windows programmed into the bridges will end up being wrong, and there is a good chance the device will be non-responsive, or worse still, will attempt to respond to addresses that some OTHER device is already set up to respond to.

Generally, attempts to do this are considered Bad Practice. There are some obsolete DDI calls that suggest that you can ask the OS to reassign addresses (such as HalAssignSlotResources and IoAssignHalResources, but it is not clear if they were actually supported properly, or even at all right now.

Just because it is *possible* to do something does not suggest that it makes sense to do so, or that doing so will not corrupt the hardware and software to the point of failure.
joe

-----Original Message-----
From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Maxim S. Shatskih
Sent: Wednesday, November 26, 2008 8:14 PM
To: Windows System Software Devs Interest List
Subject: Re:[ntdev] Question on MMIO for a PCI device

>1. where does that intelligence of mapping physical address space to device address lie?

In the BAR address decoding hardware of the device.

>2. What operations are involved (from a programmer's perspective)in accomplishing this on a PCI
>device?

PCI.SYS reads the "resource requirements" from the config space and reports them to PnP.
PnP arbitrates them - i.e. assigns unique values not clashing with anything.
PCI.SYS takes the "resources" values from PnP and programs the BARs.

>3. if I just modify the MMIO BAR of the PCI register, does it cause a different block physical address
>space to be mapped to device IO?

Yes.

--
Maxim S. Shatskih

Windows DDK MVP
xxxxx@storagecraft.com
http://www.storagecraft.com

NTDEV is sponsored by OSR

For our schedule of WDF, WDM, debugging and other seminars visit:
http://www.osr.com/seminars

To unsubscribe, visit the List Server section of OSR Online at
http://www.osronline.com/page.cfm?name=ListServer

--
This message has been scanned for viruses and
dangerous content by MailScanner, and is
believed to be clean.

anton_bassov Member MODERATED
Posts: **5,243**
November 2008

> Setting the BAR registers is a trivial operation involving I/O ports defined in the PCI
> specifications (CF8/CFC)

Actually, there is no guarantee that the values you obtain from configuration space are valid. Most of them are, but when it comes to memory and interrupt resources, there may be a mismatch between configuration space and actual settings. For example, if you read IRQs from configuration space you will get the values that correspond to PIC settings, but they are wrong for APIC HAL. Therefore, the OS has to obtain this info from BIOS tables, rather from any other source, and, if BIOS allows it (i.e. ACPI BIOS), it may modify these settings. Again, it will not always be able to do so even with ACPI BIOS, because IRQs may be physically wired together so that it will be unable to separate them....

The bottom line - as Tim and Joe said already, as a driver writer you should not mess around with it, and obtain all info that you need only from the OS (in this particular case, as translated resources that driver receives from the OS)...

Anton Bassov

Maxim_S._Shatskih Member
Posts: **10,396**
November 2008

> hard part. HalTranslateBusAddress would be responsible for creating the
> mapping from virtual address to physical address, by obtaining a block
> (perhaps only one page, perhaps many pages) of virtual address space and
> updating the memory maps to convert those virtual addresses to the physical
> addresses for the bus.

This is done by MmMapIoSpace, not by HalTranslateBusAddress.

> obsolete DDI calls that suggest that you can ask the OS to reassign
> addresses (such as HalAssignSlotResources and IoAssignHalResources, but it
> is not clear if they were actually supported properly, or even at all right
> now.

Calling any of these functions will mark the driver as "legacy" and _disable power management_ on the machine. These NT4 functions were already obsolete in year 2000.

--
Maxim S. Shatskih
Windows DDK MVP
xxxxx@storagecraft.com
http://www.storagecraft.com

OSR_Community_User Member
Posts: **110,217**

November 2008

I stand corrected. I never did a memory-mapped device or worked with one
after the fact, so had forgotten MmMapIoSpace.
joe

-----Original Message-----
From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Maxim S. Shatskih
Sent: Thursday, November 27, 2008 7:03 AM
To: Windows System Software Devs Interest List
Subject: Re:[ntdev] Question on MMIO for a PCI device

> hard part. HalTranslateBusAddress would be responsible for creating the
> mapping from virtual address to physical address, by obtaining a block
> (perhaps only one page, perhaps many pages) of virtual address space and
> updating the memory maps to convert those virtual addresses to the
physical
> addresses for the bus.

This is done by MmMapIoSpace, not by HalTranslateBusAddress.

> obsolete DDI calls that suggest that you can ask the OS to reassign
> addresses (such as HalAssignSlotResources and IoAssignHalResources, but it
> is not clear if they were actually supported properly, or even at all
right
> now.

Calling any of these functions will mark the driver as "legacy" and _disable
power management_ on the machine. These NT4 functions were already obsolete
in year 2000.

--
Maxim S. Shatskih
Windows DDK MVP
xxxxx@storagecraft.com
http://www.storagecraft.com

anton_bassov Member MODERATED
Posts: **5,243**
November 2008

> >HalTranslateBusAddress would be responsible for creating the
> > mapping from virtual address to physical address, by obtaining a block
> > (perhaps only one page, perhaps many pages) of virtual address space and
>> updating the memory maps to convert those virtual addresses to the physical
>> addresses for the bus.

> This is done by MmMapIoSpace, not by HalTranslateBusAddress.

Actually, the way I understood it, in this context by the term "virtual" Joe meant not virtual memory address but physical bus address on the machine that supports IOMMU - indeed, HalTranslateBusAddress() translates physical bus address to the physical system address that may be subsequently used in a call to

MmMapIoSpace(). Although these two are the same on x86, they may be different on architectures that support IOMMU.

BTW, please note that MmMapIoSpace() maps physical system address to the virtual address so that it can get accessed by the _CPU_ that runs in protected mode with paging enabled, i.e. it simply updates PTEs - as long as CPU runs in protected mode with paging enabled, it cannot make any use of physical system address until its gets mapped to the virtual one...

Anton Bassov

f

Maxim_S._Shatskih Member
Posts: **10,396**
November 2008

>subsequently used in a call to MmMapIoSpace(). Although these two are the same on x86, they may
>be different on architectures that support IOMMU.

I think that IOMMU is for DMA only - when the device is a master, and RAM is a slave. It does not translate the addresses if CPU is a master.

--
Maxim S. Shatskih
Windows DDK MVP
xxxxx@storagecraft.com
http://www.storagecraft.com

f

anton_bassov Member MODERATED
Posts: **5,243**
November 2008

> I think that IOMMU is for DMA only - when the device is a master, and RAM is a slave.
> It does not translate the addresses if CPU is a master.

No matter who is a bus master, all cycles turn up on the front-side bus . Even if they are issued by the device and RAM is slave they still appear on FSB. You can think of IOMMU just as of some programmable logic that sits in between FSB and the device. ...

Anton Bassov

f

Jake_Oshins Member
Posts: **1,058**
December 2008

All right. Let's work on some terminology here. Anton, you're mostly
right, but your description bears a little correction and clarification.
Joe, you're a little out of date, but others have already pointed that out.

Virtual Address: This is the address used by the processor when page tables
are enabled, which is to say all of the time when running a modern OS.
These are the addresses you see in pointers.

(System) Physical Address: This is the address actually issued on the
processor's frontside bus. This is the result of using a Virtual Address in
your code, after the processor (and/or hypervisor) has traversed all the
sets of page tables that are in use.

Guest Physical Address: This is the address perceived as physical by an OS
running on top of a hypervisor, before the last set of page tables has been
traversed.

Bus-Relative Address: This is the address as it is perceived on the I/O bus
that your device is attached to.

Device Logical Address: This is a synonym for Bus-Relative Address used in
NT DMA APIs. In general, it refers only to addresses placed on the bus as
DMA.

I/O MMU: Page translation device embedded in the memory controller/North
Bridge which translates addresses coming from devices as DMA.

Base Address Register (BAR): Register in your device which contains the Bus Relative Address (base) which your device will respond to.

Now, given this terminology, let's make some statements.

1) To access registers in MMIO space in your driver, you need a Virtual Address.

2) When your driver is "started" by the PnP manager, your driver is handed both the Physical Address base and the Bus-Relative base in your resources. The first is called a "Translated Resource List" and the second is called a "Raw Resource List" or just a "resource list." Your driver need not read its BARs.

3) When your driver is started, just before IRP_MN_START_DEVICE completes up to your FDO, the bus driver (in this case PCI) will write your Bus-Relative base addresses (from the raw resource list) into your BARs.

4) In order to get a Virtual Address to use in your code, you call MmMapIoSpace with the Translated resources. This step consumes Page Table Entries (PTEs) and so it is not automatically done for you by the PnP manager, as the PnP manager can't know if you need PTEs for every page in your BARs.

5) If you're curious about your Bus-Relative resources for some reason, they exist in your raw resource list. Again, no need to probe the BAR, certainly not by reading or writing CF8/CFC, as this can corrupt the pci driver's use of these registers.

6) DMA is another matter. If you use the NT DMA API properly, you will be handed Device Logical Addresses, which are sometimes called Scatter/Gather Lists.

7) Translations can and do exist on some machines between Physical Addresses and Bus-Relative Addresses.

8) Other translations can and do exist between Device Logical Addresses and Physical Addresses, usually because an I/O MMU is present.

9) I/O MMU translations do not affect processor-issued addresses and vice-versa. (The only exception to this was the GART in the old AGP world, which was effectively both a processor and DMA translation engine.)

10) Anton says that x86 does not involve translations between Physical Addresses and Bus-Relative Addresses. He's both right and wrong. He's right because machines with these features are becoming so rare you won't usually see them. He's wrong because this isn't a feature of the processor, and thus it's not defined by "x86." It's a feature of the memory controller in the machine. X86 can be paired with such a memory controller, and it has been in the past. This is still somewhat common in the Itanium space.

11) Virtualization adds some complexity. With Hyper-V (both in Windows Server 2008 and Windows Server 2008 R2) there is only one OS on the machine that actually sees devices. That OS also happens to be mapped such that all the Guest Physical Addresses it used happen to map without translation onto real System Physical Addresses. This is convenient, as it makes existing drivers work, even if they fail to perfectly use the NT DMA API, of if they accidentally infer a little too much from the value written into the BARs.

12) There will come a day when #11 is no longer true. At that point, it will be impossible for the Bus-Relative Addresses and the Guest Physical Address of the BARs to match. It will be extra important at that stage for drivers to pay no attention to values written into the BARs.

13) Even with a future Hyper-V (or some other hypervisor) we will probably not break the invariant we have had for the last twenty years with x86 where Device Logical Addresses equal (Guest) Physical Addresses. We might be forced to do this in some cases, though, at which point it will become much more important to correctly use the NT DMA API. In such a system, the only

drivers that would correctly function would be those that ask the OS for the
Device Logical Addresses and then use them for DMA.

--
Jake Oshins
Hyper-V I/O Architect
Windows Kernel Team

This post implies no warranties and confers no rights.

<xxxxx@hotmail.com> wrote in message news:xxxxx@ntdev...
> > >HalTranslateBusAddress would be responsible for creating the
>> > mapping from virtual address to physical address, by obtaining a block
>> > (perhaps only one page, perhaps many pages) of virtual address space
>> > and
>>> updating the memory maps to convert those virtual addresses to the
>>> physical
>>> addresses for the bus.
>
>> This is done by MmMapIoSpace, not by HalTranslateBusAddress.
>
>
> Actually, the way I understood it, in this context by the term "virtual"
> Joe meant not virtual memory address but physical bus address on the
> machine that supports IOMMU - indeed, HalTranslateBusAddress() translates
> physical bus address to the physical system address that may be
> subsequently used in a call to MmMapIoSpace(). Although these two are the
> same on x86, they may be different on architectures that support IOMMU.
>
> BTW, please note that MmMapIoSpace() maps physical system address to the
> virtual address so that it can get accessed by the _CPU_ that runs in
> protected mode with paging enabled, i.e. it simply updates PTEs - as long
> as CPU runs in protected mode with paging enabled, it cannot make any use
> of physical system address until its gets mapped to the virtual one...
>
> Anton Bassov
>

anton_bassov Member MODERATED
Posts: **5,243**
December 2008

Jake,

> Anton says that x86 does not involve translations between Physical Addresses and
> Bus-Relative Addresses. He's both right and wrong. He's right because machines with these
> features are becoming so rare you won't usually see them. He's wrong because this isn't a
> feature of the processor, and thus it's not defined by "x86."

Actually, we have a similar discussion on another thread. The thing is, CPU in itself is nothing, unless it is located on the machine with the appropriate chipset. For example, consider what happens if you plug milti-core CPU into a machine with old motherboard that does not support IOAPIC, and, hence, cannot support multiprocessing. This is why the terms like "x86", "x86_64", "arm",etc normally refer not just to a CPU but to the architecture, and CPU is just a part of it. If we continue with our above example, processor core's local APIC is just logically inseparable from IOAPIC, so that you can think of IOAPIC as of an essential part of x86 architecture, although it resides on the motherboard...

Anton Bassov

OSR_Community_User Member
Posts: **110,217**
December 2008

Thank you for such a complete summary!

-----Original Message-----
From: xxxxx@lists.osr.com
[mailto:xxxxx@lists.osr.com] On Behalf Of Jake Oshins
Sent: Monday, December 01, 2008 11:50 PM
To: Windows System Software Devs Interest List
Subject: Re:[ntdev] Question on MMIO for a PCI device

Sign In or Register to comment.

Howdy, Stranger!

It looks like you're new here. If you want to get involved, click one of these buttons!

SIGN IN REGISTER

Quick Links

Categories

Recent Discussions

Best Of...

Upcoming OSR Seminars

OSR has suspended in-person seminars due to the Covid-19 outbreak. But, **don't miss your training!** Attend via the internet instead!

Developing Minifilters	24 May 2021	Live, Online
Writing WDF Drivers	14 June 2021	Live, Online
Internals & Software Drivers	2 August 2021	Live, Online
Kernel Debugging	27 Sept 2021	Live, Online

Categories

All Categories	57.2K
Announcements and Administration	86
NTDEV	39K
NTFSD	15.3K
WINDBG	2.9K