

## Re: Handling PCI interrupts

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2008-05/msg00906.html>

---

- *From:* "David Craig" <[drivers@xxxxxxxxxx](mailto:drivers@xxxxxxxxxx)>
  - *Date:* Thu, 29 May 2008 19:08:33 -0700
- 

That makes sense but it also means the 'software interrupts' only occur on the board and are not sent to the driver. I would suggest that you determine at some time in the future when you have a close to production system which of the hardware interrupt sources is the most frequently received. Check for that interrupt first except if you have an interrupt that indicates a major failure state that requires immediate attention. If your device is a fairly slow generator of data and interrupts this is not so critical, but if you push the ability of the computer to keep the flow going, it may be critical.

Your driver will probably be in a stack of its own. Not NDIS, HID, Storage, etc. as defined by Microsoft. This gives you many options for providing the data to the client program. The simplest is to use IoCtls. First, write the program to send one, wait for it, process the data, and then repeat the sequence. Later you can consider having two or more issued to your driver using some form of sequencing so that your driver can return them in order and your program can recognize which one was completed. If the volume of data is great and occurs at high speed, you may want to consider using read and write calls to handle the data, but IoCtls do have the advantage of being able to do bidirectional transfers of information – either for the driver or device on send plus data from the driver or device in the other direction. Read/Write type logic is best for independent data transfers.

In IoCtls, the use of buffered or direct type requests are another consideration. Use buffered if the copy time required is not a factor as with transfers of less than a single 4KB page. The buffered are the easiest to use and safest from problems with worrying about accessing user buffers when they are not appropriately locked down.

"Bryan" <[Bryan@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:Bryan@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)> wrote in message <news:67ED2BA3-D1AF-407F-86B5-88C0587989FB@xxxxxxxxxxxxxxxxxxxx>

Can you explain how a hardware device has 'software interrupts'? I saw this posted twice in the last few days and it has confused me. How does a hardware device execute code in the host processor and generate a software

## Re: Handling PCI interrupts

interrupt?

The DSP has standard PCI interrupts (parity error, etc.), but in addition it also has "four software interrupts"(as the manuals calls them). I believe they are simply programmable interrupts as they are not controlled by the DSP's PCI peripheral controller. To trigger the interrupts from the device driver to the DSP, the driver just has to flip a bit in a register located in one of the memory-mapped spaces presented to the driver. Driver-to-DSP interrupts do not use PCIINT on the bus; they cause an interrupt inside the DSP's core. (This is my understanding at least)

For an application running on the DSP, it can trigger interrupts to the driver by flipping bits in a separate register. Doing so causes the DSP's PCI peripheral to generate PCIINT, and that in turns causes Windows to call all of the ISR callback routines for the PCI drivers. My driver is able to recognize that this interrupt is its responsibility by looking at that same register in the DSP (because that register is memory-mapped into the Kernel).