

[Advanced search](#)[IBM home](#) | [Products & services](#) | [Support & downloads](#) | [My account](#)[IBM : developerWorks : Linux](#) | [Open source : Linux articles](#) | [Open source articles](#)

developerWorks

Linux hardware stability guide, Part 2



Drivers, IRQs, and PCI latency

[Daniel Robbins](#) ([drobbins@gentoo.org](mailto:drobbins@gentoo.org))

President/CEO, Gentoo Technologies, Inc.

July 2001

One of Linux's claims to fame is its legendary stability. But the most stable operating system in the world won't do you any good if your hardware is defective or misconfigured. In this article, Daniel Robbins shares his experiences in getting his NVIDIA TNT graphics card working under Linux using NVIDIA's accelerated drivers. As he does, he'll show you how to diagnose and fix IRQ and PCI latency timer issues -- techniques you can use to ensure that your systems don't experience lock-ups, inconsistent behavior, or data loss.

The many causes of instability

A stability problem is often not caused by defective hardware, but by improper hardware configuration or flaky drivers. My experience in this area began when I tried to get Linux working on my Diamond Viper V550, an NVIDIA TNT-based AGP card, using NVIDIA's own accelerated drivers.

NVIDIA has their own display drivers for Linux, a collaboration between NVIDIA, SGI, and VA Linux. These drivers have a lot of advantages over the standard 2d-only NVIDIA drivers included with Xfree86 4.0. For one, they have full accelerated 3D support. Even better, they feature an official OpenGL 1.2 implementation, rather than just an enhanced version of Mesa. So, all in all, these accelerated drivers are the ones you want to be using if you own an NVIDIA-based graphics card, at least in theory. My attempt to get them working properly turned out to be an excellent learning experience, to say the least.

After I installed the accelerated Linux NVIDIA drivers (see [Resources](#) later in this article), I started up Xfree86 and began playing around with all my 3D applications, now wonderfully accelerated as they should be. Up until that point, I had had to reboot into Windows NT in order to take advantage of 3D acceleration. Now, while I don't mind NT, having to reboot to use 3D apps was somewhat annoying, and I was glad to have one less reason to leave Linux and reboot my machine. However, after playing around for an hour or so, I experienced a fatal setback to my Linux 3D aspirations -- my machine locked up. My mouse simply stopped moving and the screen froze, and I had to reboot my system.

Yes, I was having some kind of stability problem. But I didn't know exactly what was causing the problem. Did I have flaky hardware, or was the card misconfigured? Or maybe it was a problem with the driver -- did it not like my VIA KT133-based Athlon motherboard? Whatever the problem, I wanted to resolve it quickly. In this article, I'm going to share with you the procedure that I went through to fix my hardware stability problem. Although you may not be struggling with exactly the same issue, the steps that I used to diagnose and (mostly) fix the problem are general in nature and applicable to many different types of Linux hardware problems.

First, the hardware

The first thing that crossed my mind was that I might have flaky or under-cooled hardware. On the one hand, my Diamond Viper V550 seemed to have no problems under Windows NT. On the other hand, maybe Linux was somehow pushing the chip harder and triggering heat-related lock-ups. My V550 did get *extremely* hot, and its OEM heatsink seemed at best

**Contents:**[The many causes of instability](#)[First, the hardware](#)[New drivers -- and a possible solution?](#)[IRQs and PCI](#)[Fix one problem, find another](#)[The PCI latency timer](#)[PCI latency approaches](#)[Resources](#)[About the author](#)[Rate this article](#)**Related content:**[Linux hardware stability guide, Part 1](#)[Tutorial: Compiling the Linux kernel](#)[More Linux resources](#)[More Open source resources](#)

barely adequate. The combination of the lock-ups and the fact that this card was being marginally cooled convinced me to head over to PC Power and Cooling (see [Resources](#)) to purchase a mini integrated heatsink/fan for my V550.

So, I received my Video Cool, popped off the OEM heatsink on the video card (voiding the warranty), cleaned off the TNT chip and affixed the Video Cool to the top of the chip. Verdict? My video card didn't get extremely hot anymore, but the lockups continued. The lesson I learned from this particular experience is this -- if you ensure that your system is adequately cooled to begin with, you'll never need to worry about components malfunctioning due to inadequate cooling. This in itself is a good reason to invest some time and effort in making sure that your workstations and servers run coolly. Now that I had taken care of the heat issue, I knew that the lock-ups were most likely not due to flaky hardware, and I began to look elsewhere.

New drivers -- and a possible solution?

I partly suspected that NVIDIA's drivers were themselves the cause of the problem. Fortunately, a new version of the drivers had just been released, so I immediately upgraded in the hope that this would solve my stability problem. Unfortunately, it didn't, and after checking with others on the #nvidia channel on openprojects.net, I found out that while not everyone was able to get the driver to operate stably, it was working for a lot of people.

On #nvidia, someone suggested that I make sure that the V550 wasn't sharing an IRQ with another card. Unlike the standard XFree86 driver, the accelerated NVIDIA driver requires an IRQ for proper operation. To see if it had its own dedicated IRQ, I typed "cat /proc/interrupts", and lo and behold, my V550 was sharing an interrupt with my IDE controller. Before I explain how I solved this particular problem, I'd like to give you a brief background on IRQs.

PCs use IRQs, and hardware interrupts in general, to allow peripheral devices, such as the video card and the disk controllers, to signal the CPU that they have data that's ready to be processed. In the old days before the PCI bus existed, it was critical that each device in the machine had its own, dedicated IRQ. In case you are still using ISA peripherals in your machine, this is still true -- all non-PCI devices should have their own dedicated IRQ.

IRQs and PCI

However, things are a little different with the PCI bus. PCI allocates four IRQs that can be used by the PCI/AGP cards in your system. In general, these IRQs *can* be shared among multiple devices. (If you do this, make sure that all the devices doing the sharing are PCI and AGP devices.) IRQ sharing is important, especially for modern machines that may have five PCIs and one AGP slot. Without IRQ sharing, you would be unable to have more than four IRQ-using cards in your system.

There are, however, some limitations to PCI IRQ sharing. While modern motherboards BIOS and Linux kernels generally support PCI IRQ sharing, certain PCI cards may simply refuse to work properly when sharing an IRQ with another device. If you're experiencing random system lockups, especially lockups that appear to be correlated with the use of a specific hardware device, you may want to try and get all your PCI devices to use their own IRQs, just to be on the safe side. The first step is to see if any devices in your system are sharing IRQs to begin with. To do this, follow these steps:

1. Use the various hardware devices in your system, such as disk, sound, video, SCSI, etc. This ensures that Linux will handle interrupts for these various devices.
2. "cat /proc/interrupts", which will display a list and count of all interrupts which the Linux kernel has handled so far. Look in the far right column in this list. If two or more devices are listed in a single row, then they're sharing that particular IRQ.

If one of the devices in question is a non-PCI device (ISA or other legacy cards) then you've found yourself an IRQ conflict, which you can attempt to fix with your BIOS, the isapnptools package, or the physical jumpers on your peripheral cards. Note that if a device is built in to your motherboard, it is most likely a PCI device even though it doesn't occupy a physical PCI slot.

If all the devices in question are PCI or AGP devices, then whether or not you have a problem depends on your hardware. Here are some steps you can take to attempt to get all your PCI/AGP devices on their own IRQs:

1. Enter your system BIOS and disable any unused peripherals (USB, parallel port, etc.). This can free up several IRQs, giving each piece of hardware in use a greater chance of being assigned its own unique IRQs.
2. Enter the PnP section of your BIOS and make sure that your BIOS is configured for a "non-PnP" operating system. Then select the "Reset ESCD data" option. This will force your BIOS to reassign IRQs to all of your hardware devices the next time you reboot.
3. Boot Linux, use your hardware, cat /proc/interrupts and look at the result. Hopefully, all your devices are now on their own IRQs.

If your two suspect PCI devices are still sharing IRQs, you have two additional options. Some BIOS setup programs allow you to assign certain IRQs to specific PCI slots. If you have one of these rare BIOS setup programs, you can use this functionality to eliminate the conflict. If you don't have this option in your BIOS (most of us don't), there is one other sure fix for this problem -- shut down your machine, turn off the power, unplug your PC from the wall, and wait a few minutes. Then, open your system case and *physically move your PCI cards to different slots*. This option may seem a bit odd, but it will definitely work and is especially effective if you have a few free PCI slots in your system (but it may take you some time to find the correct slot for each of your cards.)

I performed the "PCI card-shuffling trick" and was able to get all the devices in my system to use a unique IRQ. Well, almost. As you can see, two of my IDE devices still shared an IRQ:

```
# cat /proc/interrupts
          CPU0
 0:   52063600          XT-PIC  timer
 1:    616810          XT-PIC  keyboard
 2:         0          XT-PIC  cascade
 5:    89084          XT-PIC  ide2, ide3
 7:   1515741          XT-PIC  eth0
 8:    155928          XT-PIC  rtc
 9:  1139761505          XT-PIC  nvidia
10:    164000          XT-PIC  Ensoniq AudioPCI
12:   4458823          XT-PIC  PS/2 Mouse
14:    664176          XT-PIC  ide0
15:    38661          XT-PIC  ide1
NMI:         0
ERR:         0
```

However, this was normal because the ide2 and ide3 devices were integrated onto the same chip on my Promise FastTrak IDE card.

So, now that (almost) all my devices had a unique IRQ, I tried my accelerated drivers and ... still experienced a lock-up in less than an hour. Apparently, the shared PCI IRQ was not the problem at all. Oh, well... time to look elsewhere (yet again).

Fix one problem, find another

After some time, I found something else that I could do to get the NVIDIA drivers to run perfectly, albeit somewhat slower -- disable AGP. As much as I didn't want to do this, the current version of the drivers permitted AGP to be turned off entirely simply by adding a single line to the XF86Config. With AGP turned off, I would reduce my video to memory bandwidth by 4x, but significantly slower 3D would still be much faster than no hardware 3D acceleration at all. After disabling AGP, I *finally* had a stable system! However, this temporary solution created another problem. Whenever 3D OpenGL animation was going on, audio playback became extremely garbled and choppy. Eek!

Fortunately, I was able to find a solution to my audio problem. I used the `setpci` utility to set more appropriate PCI bus latency timer settings for my PCI devices. I'll show you the specific solution in a minute -- but first, some background.

As you probably know, the PCI bus is a shared resource -- all your PCI cards take turns communicating over the bus, and normally, all is well. However, because the PCI bus is a shared resource with a limited (although normally adequate) bandwidth, it's possible for one PCI card to negatively affect the performance of other PCI cards in the system. For example, what happens if PCI card A is in the process of sending data across the bus and at the same moment PCI card B attempts to send data? Does card A gracefully concede use of the bus, or does it continue with its data transfer -- and if so, for how long?

The PCI latency timer

The answer to this question has everything to do with a configurable setting that each PCI device has, the PCI bus latency timer. It's normally the role of the Linux driver to set the proper PCI bus latency timer value for each PCI device in your system, and most of the time the default settings are adequate (if not optimal), all the devices get along fine and the system works properly. The PCI bus latency timer can range from zero to 248. If a device has a setting of zero, then it will immediately give up the bus if another device needs to transmit. If a device has a setting of 248, it will continue to use the

bus for a longer period of time before stopping, while the other device waits for its turn.

If all of your devices have relatively high PCI bus latency timer settings and a lot of data is being sent over the bus, then your PCI cards are generally going to have to wait *longer* before they gain control of the bus and can begin sending data. However, once they gain control of the bus they will be able to burst a lot of data across it before giving up the bus to another device. This is why high PCI bus latency timer settings *increase latency* (the delay in sending data across the bus), but also *increase effective bandwidth*. Because each device gets to burst large amounts of data across the bus without interruption, the PCI bus is used more efficiently and your PCI devices can transmit more data.

On the other hand, if all your PCI devices have low PCI bus latency settings, then they're going to gladly give up the bus if another card needs to transmit data. This results in a much lower data transmit latency, since no device is going to hold on to the bus for an extended period of time, causing other devices to wait. The dark side to all this is that low PCI bus latency timer settings *reduce* the effective PCI bus bandwidth when two or more PCI devices are operating simultaneously. This happens because large data bursts become much less frequent and control of the bus changes rapidly, increasing overhead.

Most Linux distributions include a suite of tools called `pci-utils` that allow you to view and change the latency timer settings for your PCI devices. To view your current PCI latency settings, type:

```
# lspci -v
```

Typing this command will display very detailed information about all of your PCI devices. The PCI latency setting for each device is listed on the third line, right before the IRQ setting.

#### PCI latency approaches

How does this relate to my garbled sound problem? Well, I was experiencing garbled sound because with my default PCI latency settings the way they were, the V550 dominated the PCI bus when performing 3D acceleration. Here is why. My V550 is an AGP 2X card, so when I turned off AGP (to increase stability), I reduced the card's bandwidth to main memory by 75%! Because my V550 now was trying to pump the same amount of data across the slower PCI bus, the PCI bus was nearing 100% utilization, and this was causing problems for my sound hardware. Audio devices are especially susceptible to PCI latency issues because they generally have small data buffers and need their audio data delivered to them on time to avoid buffer underruns. With my current settings, the V550 was using so much PCI bandwidth that there wasn't enough left to get data to my sound card, so I experienced audio distortion caused by buffer underruns.

There are two possible solutions to this problem. The first and most obvious solution would be to use the `setpci` command to reduce my V550's PCI latency timer. This would cause it to share the PCI bus more readily, allowing my other devices to transmit their data with less latency. I tried this solution using the `setpci` command and it worked. However, I decided not to go this route because I wanted to *maximize* my already crippled 3D graphics performance, not additionally hinder it.

I decided to try out a second higher-performance option. Instead of reducing my V550 PCI bus latency, I increased the PCI latency of all my devices to the relatively high value of 176 (devices normally default to somewhere around 32, except for my V550 which defaulted to above 200). Then I set the PCI bus latency of my latency-sensitive devices to the maximum setting, 248. This solved the problem, as I had hoped, by allowing my sound card to burst relatively huge chunks of data across the bus in one go, allowing it to maximize its use of the bus and avoid buffer-underrun conditions. At the same time, my other devices could also transmit data in chunks that were small enough not to hog the bus, but large enough to use the bus efficiently. I was particularly pleased with this solution because I was able to solve my audio problem while at the same time increasing the effective bandwidth of my development machine's PCI bus. Here's the excerpt from my system startup scripts that did the trick:

```

#"open up" the PCI bus by allowing fairly long bursts for all devices, increasing
performance
setpci -v -d *:* latency_timer=b0

#maximize latency timers for network and audio, allowing them to transmit
#more data per burst, preventing buffer over/underrun conditions

setpci -v -s 00:0f.0 latency_timer=ff
setpci -v -s 00:0e.0 latency_timer=ff

```

On the first line, the `-d *:*` option tells `setpci` to apply this setting to all PCI devices. The `latency_timer=b0` option sets the timer to 176 ("b0" is hexadecimal for 176.). the `-s` options on the last two lines specify the PCI device that will be affected by PCI bus/slot and function, rather than by vendor and device ID. These are the first numbers listed for each device when you type `lspci`. The `ff` value specifies a latency timer setting of 256, which is rounded down to 248 by `setpci`. If you're experiencing a PCI latency timer-related issue, you'll need to experiment with `lspci` and `setpci` to find the optimal values for your system. If your hardware can handle it, larger latency timer values are best.

### Resources

I hope you've found my hardware troubleshooting as good a learning experience as I did. I'm now patiently waiting for the next release of the NVIDIA drivers (0.9-7). Hopefully, these will solve my AGP-related instability issues. Below are several excellent NVIDIA-related resources that may interest you.

- Read Daniel's previous *developerWorks* article in this series, [Linux hardware stability guide, Part 1](#), where he shows you how to diagnose and fix CPU flakiness, as well as how to test your RAM for defects.
- Check out [NVIDIA's accelerated Linux drivers](#).
- If you're trying to diagnose a hardware problem related to your NVIDIA graphics card, be sure to check out the [GeForce FAQ](#). It has lots of great Linux and Windows-related information.
- For additional accelerated NVIDIA troubleshooting information, check out Christian Zander's [NVIDIA Troubleshooting Guide](#) that I posted on the gentoo.org site.
- You can get the most recent version of the troubleshooting guide and other NVIDIA-related files by using IRC and connecting to `irc.openprojects.net`. Join the `#nvidia` channel, and then `/msg ice-dcc xdcc list` to receive a list of files that you can request for automatic dcc download. This is also a great place to ask questions -- the people on `#nvidia` are generally very friendly and willing to help.
- You may want to check out the [Linux Powertweak project](#). Powertweak allows you to configure PCI latency timer settings (among other things) using a GTK and console-based interface.
- Visit [PC Power and Cooling](#) to purchase things like mini integrated heatsink/fans and Video Cool.
- Check out [Tenmax](#)'s Lasagna series of coolers, which from my experience have a more cooling capacity than Video Cool but run a little bit louder.
- Browse [more Linux resources](#) on *developerWorks*.
- Browse [more Open source resources](#) on *developerWorks*.

### About the author



Residing in Albuquerque, New Mexico, Daniel Robbins is the President/CEO of Gentoo Technologies, Inc., the creator of [Gentoo Linux](#), an advanced Linux for the PC, and the **Portage** system, a next-generation ports system for Linux. He has also served as a contributing author for the Macmillan books *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming language as well as a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at **SONY Electronic Publishing/Psygnosis**. Daniel enjoys spending time with his wife, Mary, and his new baby daughter, Hadassah. You can contact Daniel at [drobbins@gentoo.org](mailto:drobbins@gentoo.org).



---

### What do you think of this article?

Killer! (5)

Good stuff (4)

So-so; not bad (3)

Needs work (2)

Lame! (1)

### Comments?

[About IBM](#) | [Privacy](#) | [Legal](#) | [Contact](#)