How To Passthrough GPU To Your VM

PCI Device passthrough does allow you to pass over things like your GPU for proper acceleration within your Virtual Machines, which makes it possible to run games and other graphic intensive tasks with no performance loss compared to your bare-metal installed Windows for example.

A single guide cannot possibly cover all the different systems that exist. Our guide attempts to give you as much information as possible. There is a lot of information out there, and not all of it may be relevant to you and your system, do be sure to make some research of your own.

Configuring the system BIOS Important BIOS settings for getting things working properly!

- Make sure your BIOS is on it's latest update.
- Enable SVM/Virtualization Mode
- Enable VT-d
- Required for passthrough to work at all; supported CPUs.
- Enable SR-IOV
- If present.
- Disable CSM/Legacy Boot
- Enable ACS/PCI-E Access Control
- If present, set to Enabled (Auto doesn't work).
- Set PEG {NUMBER} ASPM to L0 or L0sL1
- Set whichever amount you've got to mentioned, if L0 is present, pick that.
- Enable 4G Decoding
- Disable Resizable BAR/Smart Access Memory
- You might experience 'Code 43 error' if enabled.
- Enable IOMMU
- If present, mostly for AMD boards; supported CPUs.
- Set Primary Display to CPU/iGPU
- If your CPU has an iGPU, if not skip this.
- Set Pre-Allocated Memory to 64M
- If your CPU has an iGPU, if not skip this too.
- This step is not necessary for the passthrough, but helps keeping things clean.

For ignoring some annoying "warnings" in your dmesg output, do the following

```
nano /etc/modprobe.d/kvm.conf
options kvm ignore_msrs=Y report_ignored_msrs=0
```

Hit Ctrl + X -> Y -> Enter to save changes.

Setting the Boot arguments For Intel CPUs

For GRUB; Replace the current similar line with this one in your grub file.

```
nano /etc/default/grub
```

Make these changes to that file.

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt initcall_blacklist=sysfb_init"
```

Hit Ctrl + X -> Y -> Enter to save changes.

For AMD CPUs

For GRUB; Replace the current similar line with this one in your grub file.

```
nano /etc/default/grub
```

Make these changes to that file.

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet iommu=pt initcall_blacklist=sysfb_init"
```

Hit Ctrl + X -> Y -> Enter to save changes.

Your system might not be relying on Grub, but systemd instead, this is the case when you're using ZFS. So for systemd;

```
nano /etc/kernel/cmdline
root=ZFS=rpool/ROOT/pve-1 boot=zfs intel_iommu=on iommu=pt initcall_blacklist=sysfb_init
```

Hit Ctrl + X -> Y -> Enter to save changes.

Your system might not be relying on Grub, but systemd instead, this is the case when you're using ZFS. So for systemd;

```
nano /etc/kernel/cmdline
root=ZFS=rpool/ROOT/pve-1 boot=zfs iommu=pt initcall_blacklist=sysfb_init
```

Hit Ctrl + X -> Y -> Enter to save changes.

For GRUB only Only if your system uses Grub, always required after making changes!

```
update-grub
```

For systemd only Only if your system uses systemd, always required after making changes!

```
pve-efiboot-tool refresh
```

Reboot to commit changes

```
reboot
```

Configuring IOMMU Once the Host is up and running again, we need to verify that IOMMU is enabled

For Intel CPUs

```
dmesg | grep -e DMAR -e IOMMU
```

For AMD CPUs

```
dmesg | grep -e DMAR -e IOMMU -e AMD-Vi
```

If there is no output, something is wrong. You should be seeing something like this; "DMAR: IOMMU enabled"

Enable necessary kernel modules, run the following command

```
nano /etc/modules
```

Add the following lines;

```
vfio
vfio_iommu_type1
vfio_pci
```

Hit Ctrl + X -> Y -> Enter to save changes.

After changing anything modules related, you need to refresh your initramfs.

```
update-initramfs -u -k all
```

Now check if remapping has been enabled.

```
dmesg | grep remapping
```

Should output something like this; For AMD CPUs "AMD-Vi: Interrupt remapping enabled" For Intel CPUs "DMAR-IR: Enabled IRQ remapping in x2apic mode" x2apic can be different on older CPUs, but should still work. Only if you encounter issues, consider this; You could also try adding in nox2apic to your Grub or Systemd args.

If your system doesn't support interrupt remapping, you might want to allow unsafe interrupts. Please be aware that this option might make your system unstable, but not necessarily.

```
nano /etc/modprobe.d/iommu_unsafe_interrupts.conf
```

Add the following line;

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

Hit Ctrl + X -> Y -> Enter to save changes.

Blacklisting driver modules to give the VMs full access to the graphics cards etc.

```
nano /etc/modprobe.d/pve-blacklist.conf
```

Add the following lines;

```
blacklist nouveau
blacklist nvidia
blacklist nvidiafb
blacklist snd_hda_codec_hdmi
blacklist snd_hda_intel
```

```
blacklist snd_hda_codec
blacklist snd_hda_core
blacklist radeon
blacklist amdgpu
```

Hit Ctrl + X -> Y -> Enter to save changes.

For PCI passthrough to work properly, you need to have a dedicated IOMMU group for each of the devices you want to assign to a VM. To make sure that's the case, do the following;

```
pvesh get /nodes/{nodename}/hardware/pci --pci-class-blacklist ""
```

Replace {nodename} with the name of your Proxmox node.

This is an example of one of my nodes. As you can see, my GPUs, USB Controllers, Wireless adapter has their own dedicated group.

If your devices are in the same group, be sure to double check for ACS (Access Control Service) in your BIOS. Should you not find it anywhere, you can apply an override patch by including pcie_acs_override=downstream,multifunction in your grub or systemd file, if so go back to the step where we edit our said system file.

Blacklisting your Devices Finding the corresponding IDs for your PCI devices, run the following command

```
lspci -nn | grep -i {device}
```

{device} = vga, nvidia, usb, audio, wireless, and so on. Don't include the { } brackets.

You should then see a list similar to below; "0x.00.x USB controller ... [1234:5678]" "0x:00.x VGA compatible controller ... [1234:5678]" "0x.00.x Audio Device ... [1234:5678]" Not necessary to take note of the GPU Audio ID. Only do one device reference at the time, and take note of the IDs you need.

As you can see here, the ids for my GPUs are 10de:1287 and 1002:67d. You might want to look for your USB Controller ids as well, if you encounter any issues with USB Passthrough down the line. Keep yours noted for the next step!

In general it's recommended to add the IDs for all devices you plan on using for a VM. What the blacklisting essentially does, is to block the devices for Proxmox holding them hostage, and not letting a VM have full access to it, at least in most cases.

Blacklisting the PCI device IDs for the host, run the following command

```
nano /etc/modprobe.d/vfio-pci.conf
```

Add the Device IDs in this file like this;

```
options vfio-pci ids=1234:5678,1234:5678 disable_vga=1
```

Hit Ctrl + X -> Y -> Enter to save changes.

Again, do not add the GPU Audio ID. Note that adding disable_vga=1 here will probably prevent guests from booting in SeaBIOS mode, if you use that for anything.

You can also add disable_idle_d3=1 to the end of the line, essentially disables the D3 Device Power State; which will prevent certain hardware to enter a low-power mode, that might cause issues when some are passed through, read more here. Does no harm with disabling it, might only consume more power down the line. Has shown better stability for Thunderbolt cards for example.

Now when you got all that sorted, you want to reboot your Host machine once again.

```
reboot
```

Adding your Devices to your VM(s) When your Host is up and running again, we can finally start adding in our accessories to our VMs! Before you continue, depending on your choice of GPU, AMD or Nvidia, do take a look at the bottom of the page first for making necessary configurations to your macOS.

This is totally normal with the Console window whenever a GPU is added or Display is set to none.

So, make sure you got your monitor hooked up or at least a dummy plug so that you are able to remote into the VM, with your preferred Remote App.

This is the setting you want to have set for your GPU to fully work. For some, only All Functions and ROM-Bar can be ticked. Do your own experimenting with these options!

Some cards also requires a dumped .rom file for it to work. To acquire that you must dump the file yourself from your specific card, look for the guide further down in this post.

We recently discovered that if PCI-Express is not ticked, things like HDMI-Audio might not work. But do note that your install might not boot with it ticked, then have it un-ticked.

vBIOS Dumping (optional) AMD Vendor Reset (optional) Some tweaks might be necessary depending on the type of GPU you've got, you'll need mount your EFI disk and open this file in your preferred OpenCore config editor.

AMD Users EFI/OC/config.plist

Mount your EFI drive and open the file in OpenCore Configurator. Boot-args goes under NVRAM -> 7C436110-AB2A-4BBB-A880-FE41995C9F82

AMD RX 5000 and 6000 series requires the following boot-arg to get proper output. agdpmod=pikera

Only use OpenCore Legacy Patcher if you got any GTX 600 or 700 series by Nvidia, as that series is the only ones compatible with macOS.

AMD RX 400/500 and RX 5000/6000 series does not require any patching, as they are native cards.

Nvidia Users EFI/OC/config.plist boot-args goes under NVRAM -> 7C436110-AB2A-4BBB-A880-FE41995C9F82 amfi=0x80 AMFI is enabled ngfxcompat=1 Force compat property missing ngfxgl=1 Force OpenGL property missing nvda_drv_vrl=1 nvda_drv(_vrl) variable missing

To solve the SIP error, change csr-active-config to 030A0000 Upon reboot, select Reset NVRAM from the boot-picker.

If you require any assistance with any of the above please contact us.

Revision #3
Created 2026-05-11 09:48:36 UTC by Harvey Sadler
Updated 2026-05-11 20:29:50 UTC by Harvey Sadler