

# LPC 2013

# PCI Microconference

## SR-IOV Virtual Function Lifetimes

Discussion of issues around enabling, disabling, and removing Virtual Functions

# PCIe SR-IOV Extended Capability Structure

```
% lspci -s02:00.0 -vv
```

```
02:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
```

```
...
```

```
Capabilities: [150 v1] Alternative Routing-ID Interpretation (ARI)
```

```
ARICap: MFVC- ACS-, Next Function: 1
```

```
ARICtl: MFVC- ACS-, Function Group: 0
```

```
Capabilities: [160 v1] Single Root I/O Virtualization (SR-IOV)
```

```
IOVCap: Migration-, Interrupt Message Number: 000
```

```
IOVctl: Enable- Migration- Interrupt- MSE- ARIHierarchy-
```

```
IOVSta: Migration-
```

```
Initial VFs: 8, Total VFs: 8, Number of VFs: 0, Function Dependency Link: 00
```

```
VF offset: 384, stride: 4, Device ID: 1520
```

```
Supported Page Size: 00000553, System Page Size: 00000001
```

```
Region 0: Memory at 00000000f4200000 (64-bit, prefetchable)
```

```
Region 3: Memory at 00000000f4220000 (64-bit, prefetchable)
```

```
VF Migration: offset: 00000000, BIR: 0
```

```
...
```

```
Kernel driver in use: igb
```

# Instantiating Virtual Functions (VFs)

Original, module parameter based, method of instantiating VFs:

```
% modprobe -r igb  
% modprobe igb max_vfs=7
```

Some restrictions/issues:

- May require admin access to unload/reload Physical Function (PF) driver(s) and desired VFs
- Can only supply parameters on a system wide basis (i.e. what if you have multiple `igb` devices)

## New, *sysfs* based, method [v3.8-rc1]:

<https://lkml.org/lkml/2012/12/12/262> [GIT PULL PCI changes for v3.8]

2597ba7 PCI: SRIOV control and status via *sysfs* (documentation)

bff7315 PCI: Provide method to reduce the number of total VFs supported

1789382 PCI: SRIOV control and status via *sysfs*

```
# Max (ceiling) of allowable VFs
/sys/bus/pci/devices/.../sriov_totalvfs
```

Initially set from VF configuration header. Drivers can change to a lesser value. NIC drivers typically re-allocate such that a queue corresponds to a VF.

```
# Num VFs to enable
/sys/bus/pci/devices/.../sriov_numvfs
```

```
echo -n 8 > /sys/bus/pci/devices/.../sriov_numvfs
```

Of course - which method is supported is dependent upon the driver.

*sysfs* based method (preferred going forward)

```
sriov_configure()
```

```
and possibly, pci_sriov_set_totalvfs()
```

Module Parameter based method

```
./drivers/net/ethernet/intel/igb/igb_main.c::
```

```
#ifdef CONFIG_PCI_IOV
```

```
Static unsigned int max_vfs = 0;
```

```
module_param(max_vfs, uint, 0);
```

```
MODULE_PARM_DESC(max_vfs, "Maximum ...");
```

```
#endif /* CONFIG_PCI_IOV */
```

# NumVFs and Instantiation

**Capabilities: [160 v1] Single Root I/O Virtualization (SR-IOV)**

IOVCap: Migration-, Interrupt Message Number: 000

IOVctl: Enable- Migration- Interrupt- MSE- ARIHierarchy-

IOVSta: Migration-

Initial VFs: 8, **Total VFs: 8**, **Number of VFs: 0**, Fn Dependency Link: 00

**VF offset: 384, stride: 4**, Device ID: 1520

**PF Routing ID + VF Offset + (VFn \* VF Stride)**

Creation of VFs is under the control of the PF driver and may always be special, but can we make the VFs less special afterwards?

SR-IOV Asymmetry or uniqueness

- core enumerates PF

  - call driver probe function

  - driver enables SR-IOV

  - calls core to add VFs (sort of re-enumerate)

  - call VF driver probe function

- remove VF (currently disabled via *sysfs*)

- remove PF

- core remove PF

  - unbind PF driver

  - remove VFs (reverse order)

## PF Instantiation -

```
pci_scan_root_bus
  pci_scan_child_bus
    pci_scan_slot
      pci_scan_single_device
        pci_scan_device
          pci_device_add
            device_initialize
              pci_fixup_device(header)
            pci_scan_bridge
              pci_scan_child_bus
            pci_bus_add_devices (and other callers)
          pci_bus_add_device
            device_add
```

## VF Instantiation -

```
pci_enable_sriov
  sriov_enable
    virtfn_add
      pci_alloc_dev
      pci_setup_device
      virtfn->physfn = pci_dev_get(dev)
      # acquire reference to PF

      pci_device_add
      pci_bus_add_device
```

What should we do with the VFs when the driver releases the PF?

Virtual Functions (VFs) currently must be removed by `virtfn_remove()` under the control of the Physical Function (PF) driver, not by the standard `pci_stop_and_remove_bus_device()`.

This leads to special cases, e.g., omitting the *sysfs* “remove” file for VFs.

Should we allow VFs to remain in existence when the PF's driver is detached?

SR-IOV HW

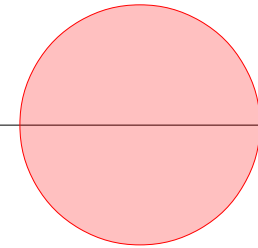
---

PF device

---

VF

---



Seems suspect, but we currently allow VFs to outlive their corresponding PF.

Seems like it would be more logical to restrict the lifetime of VFs to be  $\leq$  the lifetime of their PF (i.e. PF remove or *sysfs* deconfig).

VF

---

Should there be a way to remove an individual VF via *sysfs*?

`virtfn_remove()` function - called when disabling SR-IOV

only called for `sriov_disable` -> i.e. called by PF driver

Typically, devices have the following remove capability

```
echo -n 1 > \
/sys/bus/pci/devices/0000:bb:dd.f/remove
```

If a mechanism were put in place to disable individual VFs: Should there conversely be a way to add individual VFs?

Today there is a way to echo 1 to the remove *sysfs* entry to remove a device's `struct pci_dev`. Then a rescan will find that device again and build a new `pci_dev`.

## References:

*SR-IOV\_Primer.pdf*, PCI-SIG

*Single Root I/O Virtualization and Sharing Specification*, PCI-SIG  
(Revision 1.1, January 20, 2010)

*PCI Express Base Specification*, PCI-SIG (Revision 3.0,  
November 10, 2010): 6.13 Alternative Routing-ID Interpretation  
(ARI), 7.23 ARI Capability

*Address Translation Services*, PCI-SIG (Revision 1.1, January 26,  
2009)