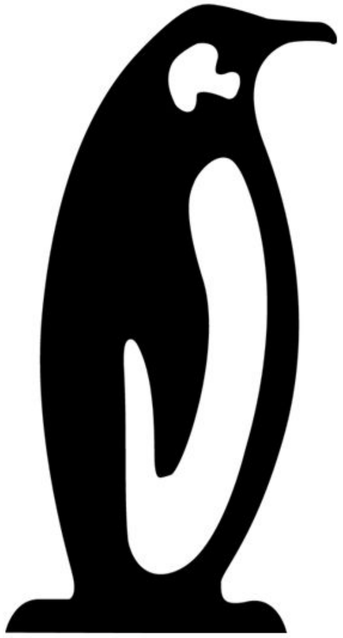

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

PCIe PortDrv - Finding a Path Forwards?

Jonathan Cameron

Too many slides (sorry), but lots of questions!

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Aim of today:
Make progress towards choosing a direction!

Motivation:
Support CXL Performance Monitoring Units on switch ports.

Thanks to all who have been involved in discussion so far: particularly Thomas Gleixner.

Background: What is drivers/pci/pcie/portdrv.c ?

- drivers/pci/pcie/*
- **PCI Driver** binding to PCI_CLASS_BRIDGE_PCI_* / PCI_CLASS_SYSTEM_RCEC
- PCIe Root ports, Switch Ports and Root Complex Event Collectors.
- Supports PCIe Spec defined features as subdrivers* on /sys/bus/pci_express
  - AER, DPC, PME, Bandwidth Notifier, Hot Plug.
- **Discovery** features, including **interrupt vectors**.

Why do we need to do anything?

- Rigid infrastructure - proving hard to extend
- Complex - perhaps more than needed:
  - Feature discovery complexity.
  - Interrupts discovery complexity.
  - Looks adaptable but isn't: Built in only, no subdriver unbind etc.
- Layering is messy: e.g.
  - AER driver provides attribute implementations used directly in drivers/pci/pci-sysfs.h
  - Interrupts - see later.

Possible conclusion: **Maybe we don't need to do anything?**

The interrupt issue

- For MSI (and MSI-X*) pci_alloc_irq_vectors() called just ‘**once**’
- **Max useful vector** must be known.
- Parent driver (portdrv) needs to know how to find that vector.
- Hence portdrv **contains interrupt vector / msgnum discovery** (as well feature discovery)

(* we’ll come back to that.)

Port Driver Probe

Set Vectors to max supported.

For each feature:
    Discover if feature present
    Discover MSI/X msgnum

Set vectors to max seen:

For each feature present
    Discover new MSI/X msgnum
    Register Device

- **Existing features:** interrupt discovery is config space.
- **Target features:** much more complex (config space to find BAR address to check etc)

Why have sub-drivers + pci_express bus for standard features?

- They aren’t really optional (no unbind!)
- Maybe we can avoid this? Options:
  1. Move into the PCIe core
  2. Make them just function calls
  3. Keep the sub-drivers

Why moving them into the core isn’t simple.

Aims:

- All the existing functionality in the PCI core
- No class driver!
- Hence for new functionality can bind a device specific driver.

Interrupts tricky - maybe we can have a disable + reenable dance? Races to close but in theory doable.

Snags:

- Kernel MSI/X infrastructure makes heavy use of devres - Device managed resources.
- Lifetime tied to driver binding.
- These can’t be in use when driver binds.
- (In theory could unwind them pre driver bind but that is a mess).
- Unsurprisingly Thomas not keen.

Worth pursuing?

So why consider move to library calls?

- Simpler - lifetimes all clear.
- Moves out the way for step 2 - I still want an extensible driver!
- Can’t use next proposal for existing features…

Maybe not a question to answer today…

Dynamic MSI-X to the rescue? (maybe)

- Dynamic MSI-X allows allocation of vectors on demand.
- Core driver can allocate interrupts it knows about.
- Subdrivers can allocate the ones they want - which core driver doesn't know about!
- Layering possible!

Limitations:

- **MSI-X only**. (currently x86 only but we can fix that)
- MSI-X msgnum sharing won't currently work with layering.

Advantages

- Clean layering as interrupt discovery belongs in child driver.
- (Pushing industry towards MSI-X)

Backup Material.
Stuff we ‘won’t’ reach!

Discovery / Unified vs Device type specific base driver?

- Discovery of extended features in core driver.
  - Can we solve that with complex ‘matching scheme’?
  - If modular subdrivers how would autoprobing work?
    - Match on DVSEC, VSEC + VID, CXL RBL, MMBRL + MCAP (6.2 feature) or custom?
- If we can’t make generic discovery work: Single driver, or ‘basic’ driver + extended drivers?
  - Basic driver PCIe Spec features only.
  - Extended driver: e.g. for CXL switch ports keeps CXL feature discovery out of basic driver.

How Dynamic MSI-X would work.

- **I have a PoC but it's still evolving.**

- Relies on per device MSI-X domains.
- Portdrv registers a hierarchical MSI-X domain on top of it's per device domain.
  - Many operations proxied to parent irq domain (the portdrv device MSI-X domain)
- Sub drivers can then request an MSI-X vector via pci_subdev_request_msi_at()

A few fiddly corners to resolve.

Thanks to Thomas Gleixner for lots of suggestions on how to make this work cleanly!

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024