CXL Upstream Intro – V2

Linaro-open-discussions 22 MAR 2021

Jonathan Cameron – Huawei Technologies R&D (UK) Ltd.

Scope of today

- CXL 2.0 specification – available from computeexpresslink.org
- PCI DOE ECN – only available to PCI-SIG members, so discussion here based on patches, not the spec.
- ACPI ECNs under code first route.
- No roadmaps – public call after all!
- Sharing current position of software support.
- Understand forthcoming problems.

Key takeaways

- CXL provides standard definition of stuff that is normally impdef.
- Typically less is left to firmware as a result.
- Current support is only a starting point!

Who is actually doing this?

- Intel team.
  - Ben Widawsky ben.widawsky@intel.com
  - Dan Williams dan.j.williams@intel.com
  - (Alison Schofield, Vishal Verma, Ira Weiny)
- DOE support from
  - Chris Browy cbrowy@avery-design.com
- Review and some emulation bits from myself and others.

What is Compute eXpress Link (CXL)?

- Lots of fancy hardware stuff – (low latency etc)
  - **Software doesn't care** (to 1st order approximation)
- Device types:
  - Type 1: Accelerator / NIC etc - coherent cache of host managed memory. (any memory on device is private to it or accessed via PCIe etc)
  - Type 2: Accelerator with memory and coherent cache of host management memory. Complex, but per device driver anyway – not much common infrastructure.
  - **Type 3: Memory Expansion**
    - **Includes switches / interleaving and other fun.**

Why is CXL memory special?

- It’s not that special…
- Discoverable
  - What’s there? (capacity, type etc)
  - What performance can we expect (latency bandwidth etc)
- Topology also discoverable (switch properties etc)
- Enough to establish NUMA characteristics

- Supports a lot of things that are IMPDEF only when dealing with DDR.
  - Hotplug
  - Hierarchical Interleaving
  - RAS features
  - Switches, including fabric management (composable systems)
- Note this stuff often wrapped up in firmware interfaces to hide that it’s implementation defined.

Why the interest now? – no CXL 1.1 products yet

- CXL 1.1 is more or less transparent to the OS.
  - **Just memory** or…
  - Devices appear as RCiEPs that needs their own drivers.
  - EDK2 – most support likely to be platform specific (no one upstreamed yet).
- CXL 2.0 is the focus
  - Getting things ready.
  - Specification prove out in a very public way ☺
    - QEMU based emulation of software interfaces.
  - We did this for CCIX as well, though stalled for various reasons.

Approach being taken

QEMU emulation

- Minimal so far
- Type 3 device
  - BAR based mailbox
  - DOE mailbox
- PCIe expansion bridge modified to support CXL. (pxb_cxl)
- CXL root ports
- No switches yet!

Kernel support

- Mailbox for device configuration
- Management interface (cdev)
- Basic sysfs description.
- Controversial bits!
  - Raw interface to userspace for commands driver doesn’t support (vendor defined or just new ones)

Trees and patches.

Mailing list: https://lore.kernel.org/linux-cxl/

Kernel

- https://gitlab.com/bwidawsk/linux/-/tree/cxl-2.0v8
- https://lore.kernel.org/linux-cxl/20210217040958.1354670-1-ben.widawsky@intel.com/T/#t

Qemu

- https://gitlab.com/bwidawsk/qemu/-/tree/cxl-2.0v4
- https://lore.kernel.org/linux-cxl/20210211185129.000055d3@Huawei.com/T/#m317aea0a3e9807fdac8a7b81fa197334fd0845ea
- Plan: https://gitlab.com/bwidawsk/qemu/-/snippets/2070304

Nodectl – userspace:

- https://lore.kernel.org/linux-cxl/20210219020331.725687-1-vishal.l.verma@intel.com/T/#t

The many mailboxes of CXL.

PCI Express config space is small; lots to describe and control so:

- Register block location structure in config space (DVSEC)
- CXL specific mailbox in PCI BAR space
  - Supports querying of available functionality.
  - (second one of these – ignore)
- PCIE ECN - Data Object Exchange Mailbox in PCIe Config Space
  - 1+ of these.
  - Slow interface mainly used for retrieving topology description CDAT.

TODO list

EDK2 support for CXL 2.0

- CXL 1.1 support like to surface with platforms
- Coldplug – it's all memory flow
- OS managed hotplug – configure HPA memory windows.

QEMU

- Emulate whatever is need to verify software stack.

Kernel

- PCI 5.0 ECNs
  - DOE mailboxes
  - CMA (component measurement and authentication)
  - IDE (integrity and data encryption)
- Hotplug
  - NUMA node hotplug.
- Switch support (CDAT via DOE etc)
- RAS
- Type 1 / type 2 devices.

TODO: Enumerating the memory

Cold-plug / firmware first

- Similar to CXL 1.1 but more generic firmware (as self describing hardware)
- OS doesn’t need to be CXL aware.
- EDK2 does the work.
- Just looks like memory
- Not clear if hotplug flows possible.

OS driven / hotplug

- Relies on preconfiguration of host.
- Fixed memory windows route to CXL RPs.
- Host interleave preset for each windows.
- OS responsible for bring up of memory. (similar to virtio-mem)

TODO: RAS flows

Error reporting via

- AER combined with..
- RAS capability in BAR space
- Event logs on devices via mailbox.

- Lot of open questions
- Likely to evolve for a while.

New stuff for 22 March

Handling of memory on Type 2 devices

- Heavily simplified – but hopefully enough for this discussion!
- For type 2 device: think GPU.
- 2 modes, tracked at a device defined granularity (lets say 4kiB)
  - Host biased – looks like a type 3 device, the coherency is managed as part of the host SoC.
  - Device biased – device has issued coherence messages to ensure there are not cached copies of our 4kiB region in the host processor. The device can then do ‘near memory’ processing.
  - Any access from Host when in Device Bias must be served (transition back to host bias)

Vikram's Question

- As type 2 memory is 'just memory' (in host biased mode) we can just use it as normal memory.
- How should it be presented by Firmware?
  - Want to be able to make it available to the GPU (driver managed) when needed only.
- Problem 1: Normal memory at boot can't be offlined later
  - Mark it as Special Purpose Memory in EFI.
  - Similar approach to the HMEM used for NVDIMMS - you deliberately hotplug later.
  - Likely to need element of BIOS control…
- Problem 2: Pinned memory in region.
  - Use ZONE_MOVEABLE.
  - Patch sets under review to migrate memory out of ZONE_MOVEABLE on pin.

Generic Initiator Reminder

- Generic initiators are first class citizen in ACPI NUMA Description.
- Originator of memory requests that is not a CPU (NIC / Accelerator etc)
- Want to be able to do clever load balancing etc, so detailed info needed by driver.
- devm_kzalloc() will allocate memory on local node or fallback to ‘nearest memory’ (SLIT)
- Linux currently initializes them as memory-less nodes.
- Same zone fallbacks etc as a CPU node that happens to have no memory.
- Simple + just works + minimal code as we need this to ‘just work’.
- Not on ARM64 this required no architecture code at all ☺

GI as bridge for CXL etc.

- UEFI code first proposal.
- Define a GI node as being both a possible initiator and target (so can be targeted by memory operations)
- Use this info to describe NUMA properties to the ‘edge’ of the SoC.
- Then use CDAT (data read from CXL end points and switches) to provide the rest of the information relevant to accessing CXL initiators and memory.
- Build kernel view of NUMA from all this info (update at runtime)

https://lore.kernel.org/linux-acpi/CAPcyv4gmd_cygXK0PpGkXmJLC3_ctEpRvpi5P-QcuXusFX5oNQ@mail.gmail.com/

Issues

- Unwanted infrastructure created (zone lists make no sense if no initiator actually in the GI node)
- Potential backwards compatibility (minor, it will just look ‘odd’).

Solutions

- New entry type in SRAT (fairly trivial to do)
- Flag to at least let OS aware of this usecase know this GI Node is not actually going to initiate anything so don’t create a ‘memory less node’ for it.

Other topics?