PCI Endpoint Subsystem Open Items

Manivannan Sadhasivam
Senior Kernel Engineer
Linaro
Agenda

- Virtio EPF drivers for Interoperability
- Devicetree Integration
- Genalloc for Outbound Window Memory Allocation
Virtio EPF drivers for Interoperability
Virtio – Overview

- Open standard for communication between drivers and devices of different types
- Initially developed by Rusty Russell
- Now maintained by a standards body
  - https://docs.oasis-open.org/virtio/virtio/
- Primarily used as an I/O virtualization framework
  - Exposing I/O devices to guests by the hypervisor
  - Used by hypervisors such as KVM, lguest, ACRN etc…
- Also used for inter chip communication within the SoC
  - RPMSG
Virtio Architecture - Simplified
PCI Endpoint Subsystem - Overview

● Used to make Linux run on PCI(e) Endpoint devices (NVMe, WLAN, Modems, etc…)

● Added as a separate subsystem/framework under PCI
  ○ drivers/pci/endpoint/

● Endpoint Controller (EPC) drivers manages the PCI transport
  ○ drivers/pci/controllers/

● Endpoint Function (EPF) drivers define the behavior of the device
  ○ drivers/pci/endpoint/functions/

● Needs equivalent drivers on the host for functionality
  ○ drivers/misc/pci_endpoint_test.c
  ○ For devices like NVMe, existing driver can be reused
    ■ drivers/nvme/host/pci.c
Virtio for PCI Endpoint Subsystem

- Idea mooted around 2019
  - https://lore.kernel.org/all/20190823213145.2016-1-haotian.wang@sifive.com/
- Existing Virtio frontend drivers on the host (Linux Kernel) can be reused
  - PCI Endpoint vendors can just develop Virtio backend drivers
- Reduces fragmentation and lead time drastically
Proposals

- Haotian Wang - 2019
Proposals

- Kishon Vijay Abraham - 2020
Proposals

- **Shunsuke Mie - 2023**
Final Implementation

- Moving forward with the proposal from Shunsuke Mie?
  - Simple yet scalable one
    - Can be extended to virtio-scsi, virtio-console, etc…
  - Involving VHOST seems be an overkill
    - Drawback of proposal 2
  - Vringh offloads the virtqueue management work
    - Drawback of proposal 1
Devicetree Integration
Problem Statement

● Binding between EPC (Endpoint Controller) and EPF (Endpoint Function) happens through ConfigFS

● No devicetree integration so far as the EPF drivers are software blocks

● But there are EPF drivers that have relevant hardware blocks

● **MHI (Modem Host Interface)** on Qualcomm chipsets is one example
  ○ MHI is a Qualcomm specific protocol using PCI as the physical layer
  ○ Used for transferring data packets between PCI host and endpoint
  ○ MHI has a hardware implementation in Qualcomm chipsets supporting PCI Endpoint mode
  ○ Currently, PCI Endpoint Controller (EPC) devicetree node is used for fetching EPF specific resources like BAR region, interrupt etc...
Proposal

● A devicetree node for MHI function
  ○ Child node of PCI Endpoint Controller (EPC) node

● EPF device will be created for each function and bound with EPF driver

● Properties
  ○ reg
  ○ function-name
  ○ bar-regions
  ○ Interrupts

● Linking between EPC and EPF possible without ConfigFS
Devicetree Binding

```c
soc {
  ...

  pcel_ep: pcie-ep@1c08000 {
    compatible = "qcom,sm8450-pcie-ep";
    #address-cells = <1>;
    #size-cells = <0>;
    ...

    func@0 {
      reg = <0>;
      function-name = "sdx55-mhi";
      bar-regions = /bits/ 64 <0x01c0b000 0x1000>;
      interrupts = <GIC_SPI 440 IRQ_TYPE_LEVEL_HIGH>;
    }
  }
};
```
Genalloc for Outbound Window Memory Allocation
Problem Statement

- PCI Endpoint subsystem uses a custom memory allocator
  - drivers/pci/endpoint/pci-epc-mem.c
- Works well, but defeats the purpose of “Genalloc/Genpool” framework
Proposal

- Adapt “Genalloc/Genpool” framework for Endpoint subsystem
- Use existing “addr_space” region defined in EPC devicetree node
  - Backwards compatible with current allocator
Questions?