DPDK

Method for sharing a (PCI) device between multiple PMDs

Fiona Trahe

DPDK Summit Userspace - Dublin- 2017

Problem Statement

- Hardware accelerators can provide multiple functions via the same PCI device

  e.g. Intel's QuickAssist devices provide symmetric crypto, asymmetric crypto and compression functions under the same BAR.

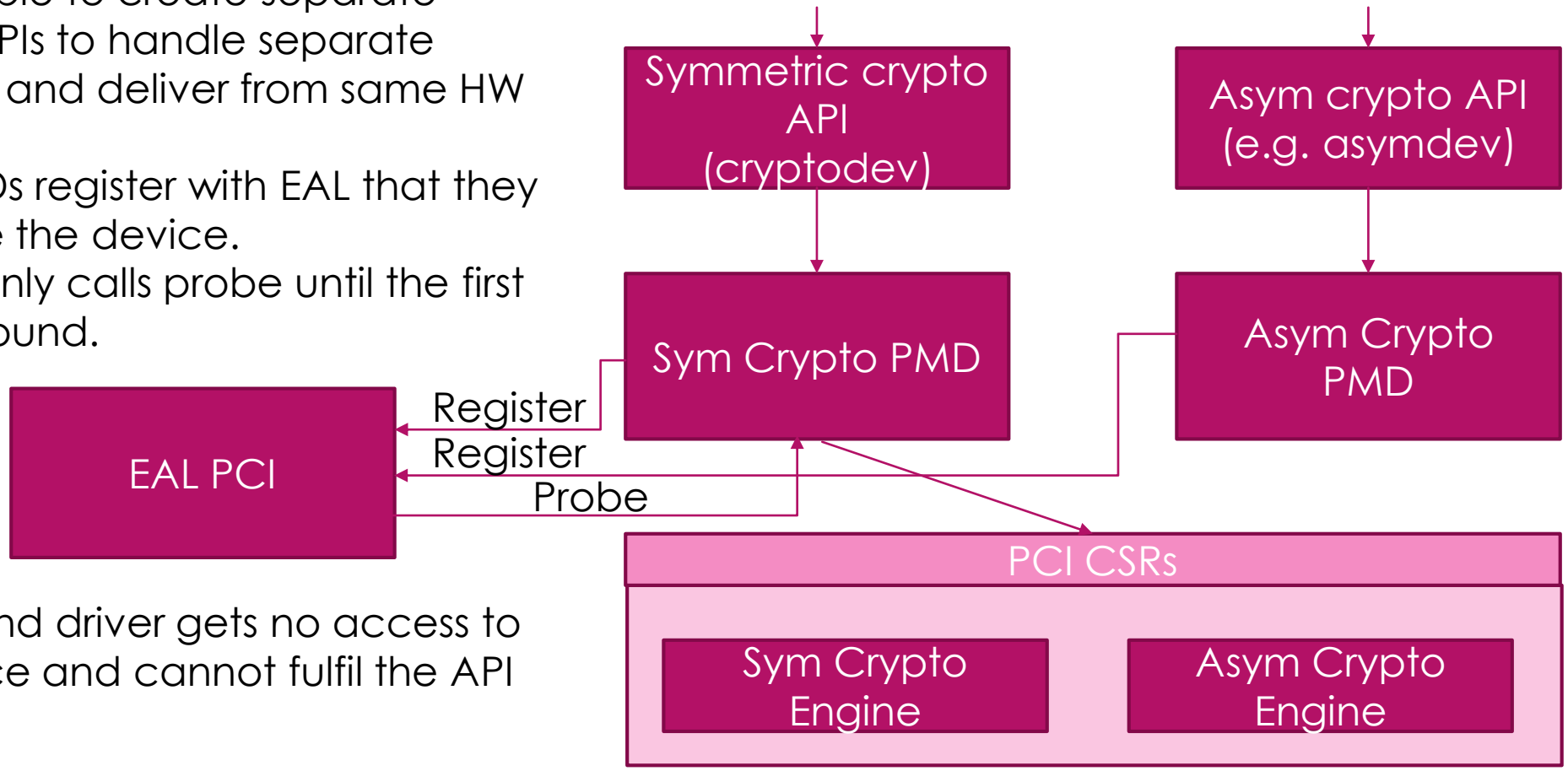
- EAL PCI presumes one and only one driver per device. This can lead to artificially bundling multiple functions under one API though it may be desirable to present these to applications using separate APIs.
- This presentation proposes a mechanism to share a pci device between multiple PMDs.
- It may also be extendable to non-pci devices.

Example problem

- Using Symmetric and Asymmetric crypto as examples of independent functions in the following diagrams.
- Just to illustrate the generic problem.
- Doesn’t imply that these APIs shouldn’t be grouped under the same API, as currently proposed in Asymmetric RFC.

Limitation: 1 PCI device – 1 driver

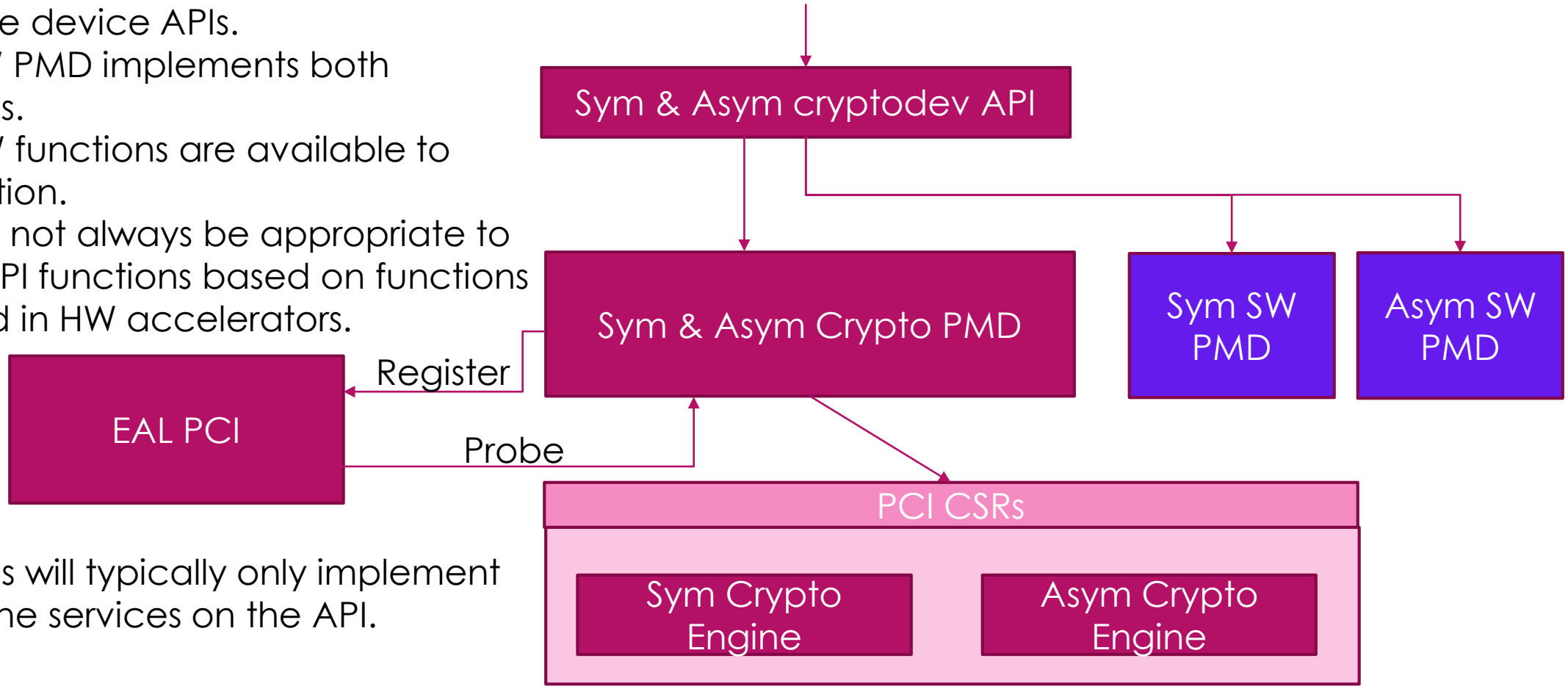
- Not possible to create separate device APIs to handle separate functions and deliver from same HW device.
- Both PMDs register with EAL that they can drive the device.
- The PCI only calls probe until the first driver is found.



- The second driver gets no access to the device and cannot fulfil the API

Option1 – combine APIs

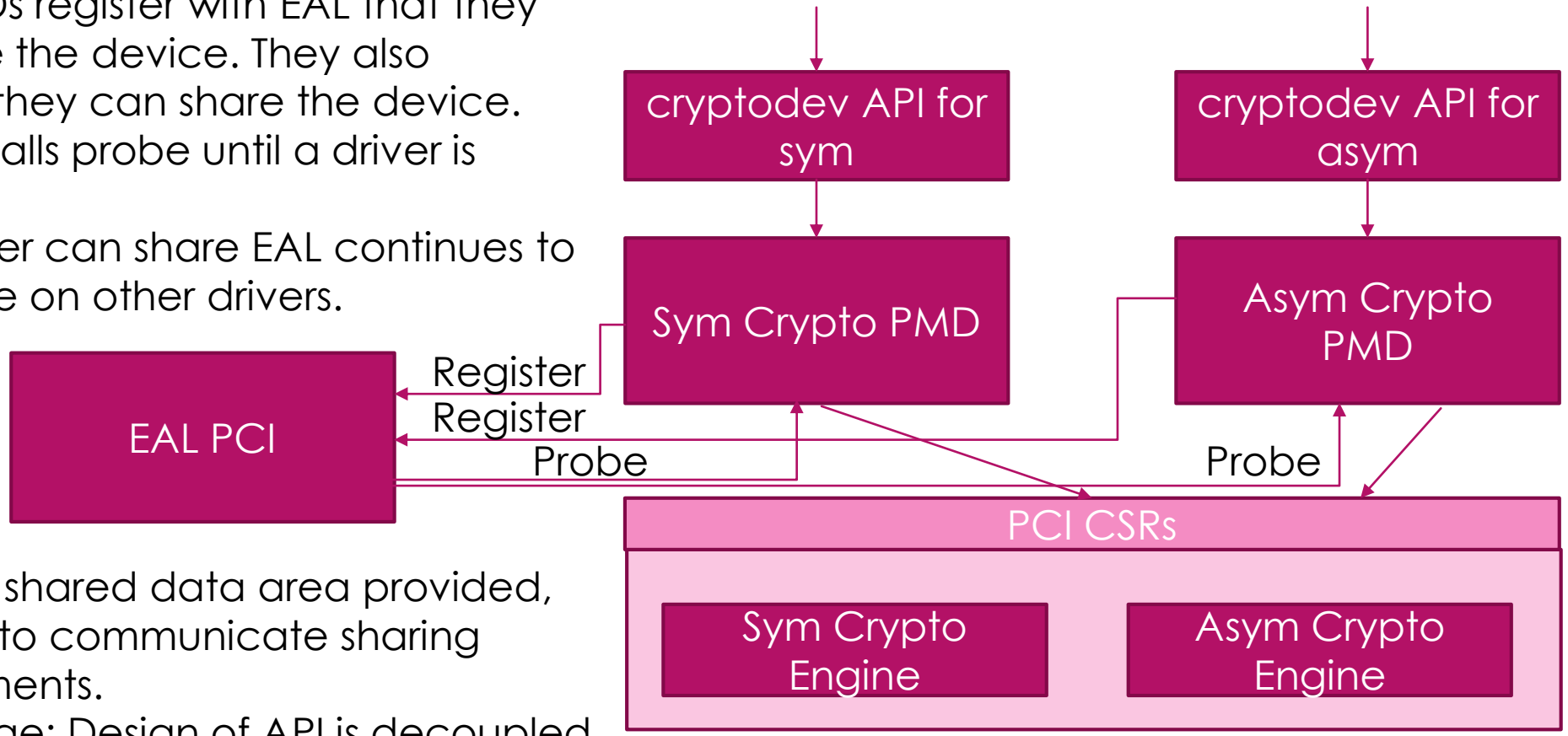
- Both functions are combined under the same device APIs.
- One HW PMD implements both functions.
- Both HW functions are available to application.
- But may not always be appropriate to group API functions based on functions grouped in HW accelerators.



- SW PMDs will typically only implement one of the services on the API.

Option2 – allow PMDs to share devices

- Both PMDs register with EAL that they can drive the device. They also indicate they can share the device.
- The PCI calls probe until a driver is found.
- If this driver can share EAL continues to call probe on other drivers.



- There's a shared data area provided, for PMDs to communicate sharing arrangements.
- Advantage: Design of API is decoupled from HW design.

Shared data area

- Up to PMDs to decide how they can share. Not prescriptive, it will vary depending on HW device design.
- For example it could contain
  - a set of resources. Each driver marks which resources it’s using.
  - locks to control access to shared CSRs and shared data.
  - state, e.g. if some part of device setup should only be done once, then could be done by first PMD and marked as done.

Status of development

- POC working with 2 QAT PMDs, the sym crypto driver and another dummy driver.
- Open issue re how ethdev drivers can accommodate the changes.

Some of the tricky bits:

- Cryptodev used BDF string as name, needed to be unique, e.g. name = “0000:02:01.1”

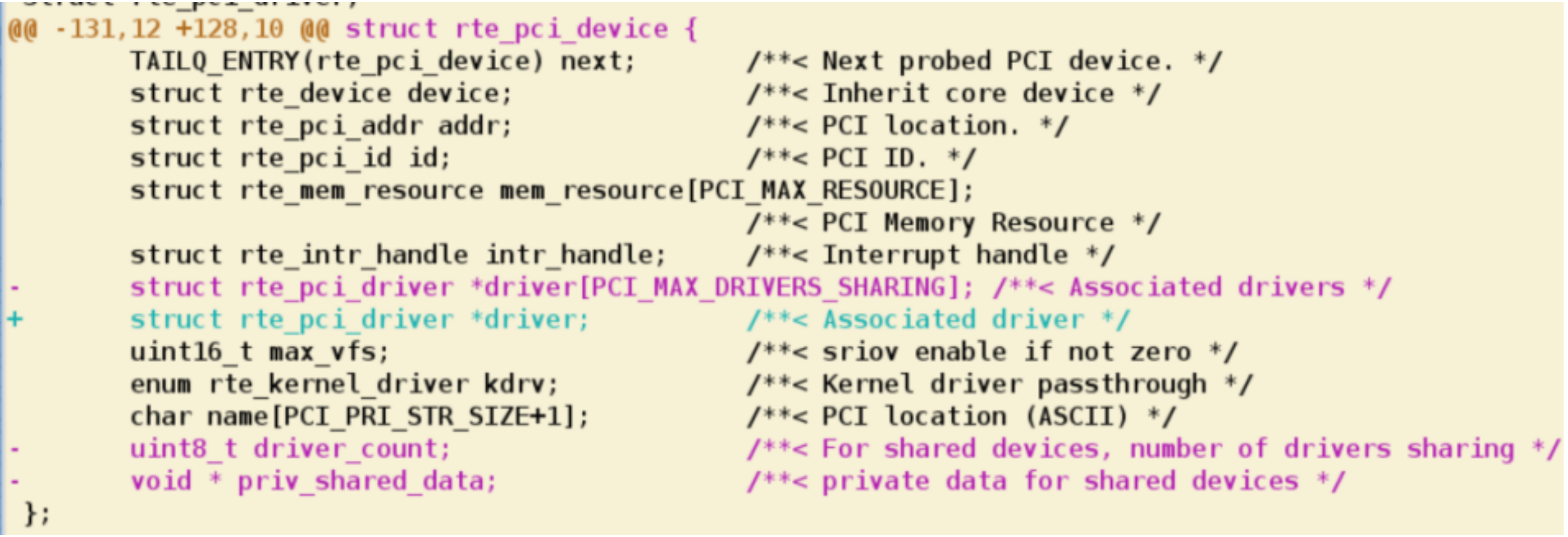
Fixed by adding function to device name e.g. “0000:02.01.1.sym” and “0000:02.01.1.asym”.

- Shareability added as driver attribute rather than device attribute. I.e. indicates driver understands how to share devices. It’s not a promise that it will share them. It could mark in the shared data area that it has taken all the resources and won’t share. If a second driver needs resources already taken by the first it will only be probed if it’s capable of sharing and should fail to start due to lack of resources.
- rte_pci_devices are in global table. Each contains rte_driver and rte_pci_driver ptrs. Written by first probe, overwritten by second. Fix: Replaced rte_pci_driver with an array. Repercussions in ethdev need investigation. Still investigating rte_driver usage.

What it's not

- Not a mechanism for sharing a device across processes. Shares a device across PMDs within a process NOT across processes.
- Not a mechanism for making all devices on a bus shareable. The driver decides whether it's capable of sharing a specific device-type with another driver.
- Not a framework for handling how the drivers share the device. The details of how they share it are controlled by the PMDs, not part of this mechanism. Supporting functions, e.g. like a regmap library for managing access to shared CSRs could be added later.

Code snippet



```
@@ -131,12 +128,10 @@ struct rte_pci_device {
        TAILQ_ENTRY(rte_pci_device) next;       /**< Next probed PCI device. */
        struct rte_device device;               /**< Inherit core device */
        struct rte_pci_addr addr;               /**< PCI location. */
        struct rte_pci_id id;                   /**< PCI ID. */
        struct rte_mem_resource mem_resource[PCI_MAX_RESOURCE];
                                                /**< PCI Memory Resource */
        struct rte_intr_handle intr_handle;     /**< Interrupt handle */
-       struct rte_pci_driver *driver[PCI_MAX_DRIVERS_SHARING]; /**< Associated drivers */
+       struct rte_pci_driver *driver;          /**< Associated driver */
        uint16_t max_vfs;                       /**< sriov enable if not zero */
        enum rte_kernel_driver kdrv;            /**< Kernel driver passthrough */
        char name[PCI_PRI_STR_SIZE+1];          /**< PCI location (ASCII) */
-       uint8_t driver_count;                   /**< For shared devices, number of drivers sharing */
-       void * priv_shared_data;                /**< private data for shared devices */
 };
```

Code snippets


```
static struct rte_pci_driver rte_qat_pmd = {
        .id_table = pci_id_qat_map,
-       .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_CAN_SHARE,
+       .drv_flags = RTE_PCI_DRV_NEED_MAPPING,
        .probe = crypto_qat_pci_probe,
        .remove = crypto_qat_pci_remove
};
```

```
        FOREACH_DRIVER_ON_PCIBUS(dr) {
                rc = rte_pci_probe_one_driver(dr, dev);
                if (rc < 0)
                        /* negative value is an error */
                        return -1;
                if (rc > 0)
                        /* positive value means driver doesn't support it */
                        continue;
+               /* driver has claimed this device.
+                * Check if it can share it with other PMDs, if so continue searching.*/
+               if (dr->drv_flags & RTE_PCI_DRV_CAN_SHARE)
+                       continue;
                return 0;
        }
```

Questions?

Feedback?

Fiona Trahe

fiona.trahe@intel.com