DEVICE TYPE AGNOSTIC DPDK: AN UPDATE

Hemant Agrawal, Shreyansh Jain

April-2017

NEXT ~12 MIN

- Overview of Bus-Device-Driver Model
- NXP Roadmap

NEXT ~10 MIN

- Overview of Bus-Device-Driver Model
- NXP Roadmap

Pre-16.11 Device⇔Driver Model

- DPDK was an inherently PCI inclined model
  - Core (EAL) libraries considered PCI objects as a first-class member

```
struct rte_eth_dev {
      eth_rx_burst_t rx_pkt_burst;
      eth_tx_burst_t tx_pkt_burst;
      ...
      struct rte_pci_device *pci_dev;
```

```
struct rte_cryptodev {
      dequeue_pkt_burst_t dequeue_burst;
      enqueue_pkt_burst_t enqueue_burst;
      ...
      struct rte_pci_device *pci_dev;
```

```
struct eth_driver {
      struct rte_pci_driver pci_drv;
      ...
```

  - PCI bus scan, probing, naming – all were part of `librte_eal`

```
int rte_eal_init(int argc, char **argv)
{
      ...
      rte_eal_pci_init()
      ...
      rte_eal_pci_probe()
```

Pre-16.11 Device⇔Driver Model

- DPDK was an inherently PCI inclined model
  - Core (EAL) libraries considered PCI objects as a first-class member

```
struct rte_eth_dev {
      eth_rx_burst_t rx_pkt_burst;
      eth_tx_burst_t tx_pkt_burst;
      ...
      struct rte_pci_device *pci_dev;
```

```
struct rte_cryptodev {
      dequeue_pkt_burst_t dequeue_burst;
      enqueue_pkt_burst_t enqueue_burst;
      ...
      struct rte_pci_device *pci_dev;
```

```
struct eth_driver {
      struct rte_pci_driver pci_drv;
      ...
```

  - PCI bus scan, probing, naming – all were part of `librte_eal`

```
int rte_eal_init(int argc, char **argv)
{
      ...
      rte_eal_pci_init()
      ...
      rte_eal_pci_probe()
```

- Mempool handlers part of `librte_mempool` core library
  - New handler (hardware backed) meant changing the library

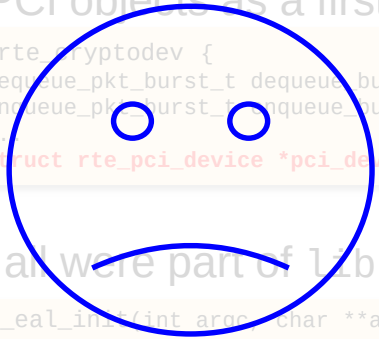
Pre-16.11 Device⇔Driver Model

- DPDK was an inherently PCI inclined model
  - Core (RTE) libraries considered PCI objects as a first-class member

```
struct rte_eth_dev {
        eth_rx_burst_t rx_pkt_burst;
        eth_tx_burst_t tx_pkt_burst;
        ...
        struct rte_pci_device *pci_dev;
```

```
struct rte_cryptodev {
        dequeue_pkt_burst_t dequeue_burst;
        enqueue_pkt_burst_t enqueue_burst;
        ...
        struct rte_pci_device *pci_dev;
```

```
struct eth_driver {
        struct rte_pci_driver pci_drv;
        ...
```



  - PCI bus scan, probing, naming – all were part of `librte_eal`

```
int rte_eal_init(int argc, char **argv)
{
        ...
        rte_eal_pci_init()
        ...
```

Without changing EAL, adding a new set of `rte_xxx_device/rte_xxx_driver`, was not possible.
(Or, of course, spin your own DPDK)

Core library changes are not easy – for a maintainer, as well as community.
They impact everyone irrespective of their size – need to 'handle' impact across all supported devices

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

  All tied together through EAL
- NXP has published its drivers for above three constructs:
  - FSLMC Bus driver
  - DPAA2 hardware based mempool driver
  - DPAA2 Poll Mode Driver

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

```
struct rte_bus {
      TAILQ_ENTRY(rte_bus) next;
      const char *name;
      rte_bus_scan_t scan;
      rte_bus_probe_t probe;
```

```
rte_bus_register(struct rte_bus *bus);
rte_bus_unregister(struct rte_bus *bus);

RTE_REGISTER_BUS(nm, bus)
```

Global Bus list for all buses registered with EAL:

```
TAILQ_HEAD(rte_bus_list, rte_bus);
```

- An example Bus ‘driver’

```
void rte_fslmc_driver_register(struct rte_fslmc_driver *driver);
void rte_fslmc_driver_unregister(struct rte_fslmc_driver *driver);

struct rte_fslmc_bus rte_fslmc_bus = {
        .bus = {
                .scan = rte_fslmc_scan,
                .probe = rte_fslmc_probe,
        },
        .device_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.device_list),
        .driver_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.driver_list),
};

RTE_REGISTER_BUS(FSLMC_BUS_NAME, rte_fslmc_bus.bus);
```

Through `RTE_PMD_REGISTER_DPAA2(...)` Constructor, initiated from DPAA2 PMDs

Local list of Devices registered with the Bus:

```
TAILQ_HEAD(rte_fslmc_device_list,
rte_fslmc_device);
```

Local list of Drivers registered with the Bus:

```
TAILQ_HEAD(rte_fslmc_driver_list,
rte_fslmc_driver);
```

- `rte_eal_init` calls scan/probe for all registered buses - serially

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

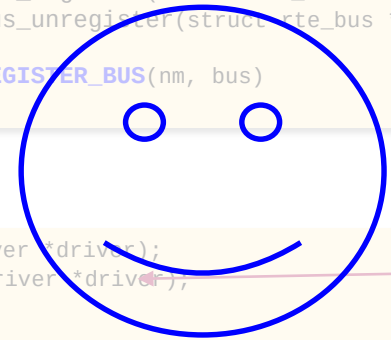
```
struct rte_bus {
      TAILQ_ENTRY(rte_bus) next;
      const char *name;
      rte_bus_scan_t scan;
      rte_bus_probe_t probe;
```

```
rte_bus_register(struct rte_bus *bus);
rte_bus_unregister(struct rte_bus *bus);

RTE_REGISTER_BUS(nm, bus)
```

Global Bus list for all buses registered with EAL:

```
TAILQ_HEAD(rte_bus_list, rte_bus);
```



- An example Bus ‘driver’

```
void rte_fslmc_driver_register(struct rte_fslmc_driver *driver);
void rte_fslmc_driver_unregister(struct rte_fslmc_driver *driver);

struct rte_fslmc_bus rte_fslmc_bus = {
        .bus = {
                .scan = rte_fslmc_scan,
                .probe = rte_fslmc_probe,
        },
        .device_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.device_list),
        .driver_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.driver_list),
};

RTE_REGISTER_BUS(FSLMC_BUS_NAME, rte_fslmc_bus.bus);
```

Through
`RTE_PMD_REGISTER_fslmc(...)`
Constructor, initiated from DPAA2 PMDs

Local list of Devices registered with the Bus:

```
TAILQ_HEAD(rte_fslmc_device_list,
rte_fslmc_device);
```

Local list of Drivers registered with the Bus:

```
TAILQ_HEAD(rte_fslmc_driver_list,
rte_fslmc_driver);
```

- Everything placed with ‘`drivers/bus/fslmc`’ folder. No changes in EAL!

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

```
struct rte_mempool_ops {
      char name[RTE_MEMPOOL_OPS_NAMESIZE]
      rte_mempool_alloc_t alloc;
      rte_mempool_free_t free;
      rte_mempool_enqueue_t enqueue;
      rte_mempool_dequeue_t dequeue;
      rte_mempool_get_count get_count;
```

```
struct rte_mempool_ops_table {
      rte_spinlock_t sl;
      uint32_t num_ops;
      struct rte_mempool_ops ops[...]

#define MEMPOOL_REGISTER_OPS(ops)
```

Global array of all Mempool handlers registered with EAL:

```
struct rte_mempool_ops_table
rte_mempool_ops_table
```

- An example Mempool ‘driver’

```
static struct rte_mempool_ops dpaa2_mpool_ops = {
        .name = "dpaa2",
        .alloc = rte_hw_mbuf_create_pool,
        .free = rte_hw_mbuf_free_pool,
        .enqueue = rte_hw_mbuf_free_bulk,
        .dequeue = rte_hw_mbuf_alloc_bulk,
        .get_count = rte_hw_mbuf_get_count,
};

MEMPOOL_REGISTER_OPS(dpaa2_mpool_ops);
```

Default Mempool controlled through configuration option:

```
CONFIG_RTE_MBUF_DEFAULT_MEMPOOL_OPS=
"dpaa2"
```

And not limited to this. Can be explicitly selected through combination of **`rte_mempool_create_empty`** and **`rte_mempool_set_ops_byname`**

- APIs exposed by EAL for mempool create/destroy/enqueue/dequeue

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

```
struct rte_mempool_ops {
      char name[RTE_MEMPOOL_OPS_NAMESIZE]
      rte_mempool_alloc_t alloc;
      rte_mempool_free_t free;
      rte_mempool_enqueue_t enqueue;
      rte_mempool_dequeue_t dequeue;
      rte_mempool_get_count get_count;
```

```
struct rte_mempool_ops_table {
      rte_spinlock_t sl;
      uint32_t num_ops;
      struct rte_mempool_ops ops[...]

#define MEMPOOL_REGISTER_OPS(ops)
```

Global array of all Mempool handlers registered with EAL:

`struct rte_mempool_ops_table rte_mempool_ops_table`

- An example Mempool ‘driver’

```
static struct rte_mempool_ops dpaa2_mpool_ops = {
        .name = "dpaa2",
        .alloc = rte_hw_mbuf_create_pool,
        .free = rte_hw_mbuf_free_pool,
        .enqueue = rte_hw_mbuf_free_bulk,
        .dequeue = rte_hw_mbuf_alloc_bulk,
        .get_count = rte_hw_mbuf_get_count,
};

MEMPOOL_REGISTER_OPS(dpaa2_mpool_ops);
```

Default Mempool controlled through configuration option:

`CONFIG_RTE_MBUF_DEFAULT_MEMPOOL_OPS= “dpaa2“`

And not limited to this. Can be explicitly selected through combination of **`rte_mempool_create_empty`** and **`rte_mempool_set_ops_byname`**

- Everything placed with ‘`drivers/mempool/dpaa2`’ folder. No changes in EAL!

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

```
struct rte_dpaa2_driver {
      TAILQ_ENTRY(rte_dpaa2_driver) next;
      rte_dpaa2_probe_t probe;
      rte_dpaa2_remove_t remove;
      struct rte_driver driver;
      ...

struct rte_dpaa2_device {
      TAILQ_ENTRY(rte_dpaa2_device) next;
      struct rte_device device;
      ...
```

Registering driver with Bus:

```
RTE_PMD_REGISTER_DPAA2(net_dpaa2,
&rte_dpaa2_pmd)
...

rte_dpaa2_driver rte_dpaa2_pmd = {
      .probe = rte_dpaa2_probe,
      .remove = rte_dpaa2_remove,
```

Ethernet instance of Device:

```
rte_dpaa2_probe(...) {
    rte_eth_dev_allocate(name);
    ...
    eth_dev->dev_ops= &dpaa2_ethdev_ops;

...
eth_dev_ops dpaa2_ethdev_ops {
      .dev_configure    = .._configure,
      .dev_start        = .._dev_start,
      .dev_stop         = .._dev_stop,
      .dev_close        = .._dev_close,
```

- What changed from 16.07…
  - `rte_eth_dev_pci_generic_probe` from `librte_ether` or own implementation of `rte_XXX_driver.probe`; Similarly for `rte_eth_dev_pci_generic_remove`
  - `rte_eal_init` now calls `rte_bus_scan()` and `rte_bus_probe()`
    - Bus operations scan over all registered buses; scanning for devices on a bus; probing for devices and attaching drivers registered on the bus.

16.11 and beyond…

- Three major constructs: Bus, Pool, Drivers (Net, Crypto)

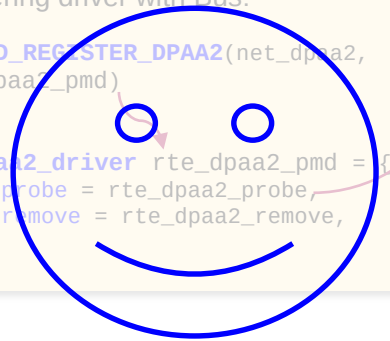
```
struct rte_dpaa2_driver {
      TAILQ_ENTRY(rte_dpaa2_driver) next;
      rte_dpaa2_probe_t probe;
      rte_dpaa2_remove_t remove;
      struct rte_driver driver;
      ...

struct rte_dpaa2_device {
      TAILQ_ENTRY(rte_dpaa2_device) next;
      struct rte_device device;
      ...
```

Registering driver with Bus:

```
RTE_PMD_REGISTER_DPAA2(net_dpaa2,
&rte_dpaa2_pmd)
...

rte_dpaa2_driver rte_dpaa2_pmd = {
      .probe = rte_dpaa2_probe,
      .remove = rte_dpaa2_remove,
```



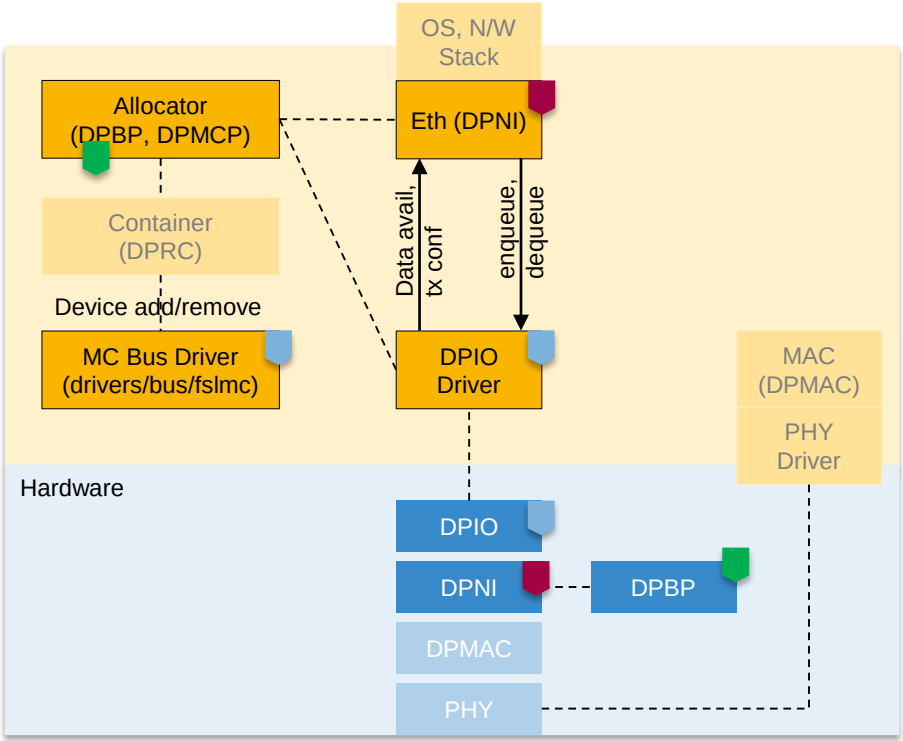
Ethernet instance of Device:

```
rte_dpaa2_probe(...) {
    rte_eth_dev_allocate(name);
    ...
    eth_dev->dev_ops= &dpaa2_ethdev_ops;

...
eth_dev_ops dpaa2_ethdev_ops {
      .dev_configure    = .._configure,
      .dev_start        = .._dev_start,
      .dev_stop         = .._dev_stop,
      .dev_close        = .._dev_close,
```

- What changed from 16.07…
  - `rte_eth_dev_pci_generic_probe` from `librte_ether` or own implementation of `rte_XXX_driver.probe`; Similarly for `rte_eth_dev_pci_generic_remove`
  - `rte_eal_init` now calls `rte_bus_scan()` and `rte_bus_probe()`
    - Bus operations scan over all registered buses; scanning for devices on a bus; probing for devices and attaching drivers registered on the bus.
- Everything placed with '`drivers/net`' and '`drivers/crypto`' folder. As usual!

DPAA2 Architecture – DPDK Layout



Derived from: https://www.kernel.org/doc/readme/drivers-staging-fsl-mc-README.txt

- `/drivers/net/dpaa2`
- `/drivers/bus/fslmc`
- `/drivers/mempool/dpaa2`

NXP's implementation for FSLMC Bus, DPAA2 Hardware Mempool and PMD are available in 17.05-rc2

NEXT ~2 MIN

- Overview of Bus-Device-Driver Model
- NXP Roadmap

NXP Roadmap

- Coming soon…
  - NXP's Bus, hardware Mempool and PMD (net and crypto) are available in 17.05-rc2
  - And hopefully these would also make it to DPDK 17.05
- Next…
  - Event Driver Framework
  - QoS Framework
  - Support for Flow Director

SECURE CONNECTIONS
FOR A SMARTER WORLD