Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2017 Intel Corporation.
Who am I

- Software Engineer in Intel and a DPDK developer
- DPDK next-net maintainer
Agenda

- What is the bus infrastructure in DPDK?
- History of the rte_bus.

Hard to resist
Quick History

v16.07
Initial approach for new bus

PCI and VDEV in EAL

v16.11
rte_bus abstraction

v17.02
Register PCI and VDEV buses
drivers/bus: fslmc

v17.05
Extend rte_bus for hotplug
devargs rework
Move PCI and VDEV to drivers/bus/

drivers/bus: dpaa

v17.08
v17.11
Where is the bus here?
Nowhere! Only PMDs should know about the bus, internal to DPDK.
What is the bus infrastructure in DPDK

- It doesn’t drive bus controllers. DPDK is not Linux.
- It is for logically grouping devices.
- Bus infrastructure responsibilities:
  - Scan devices on given bus.
  - Match device – drivers on given bus.
  - Plug / unplug a device on given bus.
- It enables creating helper functions for PMDs.
Bus scope and expectations

- Adding new buses should be easy
- Adding a new bus should not effect the core EAL code
- Bus specific code should be moved from EAL to bus

- Functional device layers should be bus agnostic
- Adding new bus types should not require change in ethdev
- Bus related information should be saved in ethdev in a generic way

- PMDs knows about the bus
- PMDs need information from EAL related to bus
- Common tasks on given bus should be easy to do for PMDs
Bus process flow (as 17.08)

**constructor**
- `rte_bus_register` → INSERT to `rte_bus_list`
- `driver_register` → INSERT to `xxx_bus.driver_list`
- `driver->bus = xxx_bus`

**eal_init**
- **FOREACH bus** → `bus->scan()`
- **FOREACH bus** → `bus->probe()`
- **FOREACH device** → `bus_common_tasks()`
- PMD specific probe()
Related data structures (as 17.08)

- rte_pci_device
- rte_pci_bus
- rte_bus
- rte_pci_driver
- rte_eth_device
- rte_device
- rte_driver
struct rte_device {
    TAILQ_ENTRY(rte_device) next; /**< Next device */
    const struct rte_driver *driver; /**< Associated driver */
    int numa_node; /**< NUMA node connection */
    struct rte_devargs *devargs; /**< Device user arguments */
};

struct rte_driver {
    TAILQ_ENTRY(rte_driver) next; /**< Next in list. */
    const char *name; /**< Driver name. */
    const char *alias; /**< Driver alias. */
};

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; /**< Associated driver */
    uint16_t max_vfs; /**< sriov enable if not zero */
    enum rte_kernel_driver kdrv; /**< Kernel driver */
};

struct rte_pci_driver {
    TAILQ_ENTRY(rte_pci_driver) next; /**< Next in list. */
    struct rte_driver driver; /**< Inherit core driver. */
    pci_probe_t *probe; /**< Device Probe function. */
    pci_remove_t *remove; /**< Device Remove function. */
    const struct rte_pci_id *id_table; /**< ID table, NULL terminated. */
    uint8_t attached; /**< Flag indicating the port is attached */
};

struct rte_vdev_driver {
    TAILQ_ENTRY(rte_vdev_driver) next; /**< Next in list. */
    struct rte_vdev_device *vdev; /**< Virtual device */
    const struct rte_driver *driver; /**< Driver for this device */
    struct rte_vdev_probe_t *probe; /**< Virtual device probe function. */
    struct rte_vdev_remove_t *remove; /**< Virtual device remove function. */
    uint8_t attached; /**< Flag indicating the port is attached */
};

struct rte_eth_dev {
    const struct eth_driver *driver; /**< Driver for this device */
    struct rte_pci_device *pci_dev; /**< PCI info. supplied by probing */
    uint8_t attached; /**< Flag indicating the port is attached */
};

struct eth_driver {
    struct rte_pci_driver pci_drv; /**< The PMD is also a PCI driver. */
    eth_dev_init_t *eth_dev_init; /**< Device init function. */
    eth_dev_uninit_t *eth_dev_uninit; /**< Device uninit function. */
    unsigned int dev_private_size; /**< Size of device private data. */
};
Bus functionalities hardcoded

16.11-17.02 -17.05-17.08-17.11

**eal init flow**

```
rte_eal_init
  eal_parse_args
  eal_parse_common_option
  rte_eal_devargs_add
  insert(devargs_list, devargs)
  rte_eal_pci_init
  rte_eal_pci_scan
  insert(pci_device_list, dev)
  rte_eal_dev_init
  foreach(devargs, devargs_list) rte_eal_vdev_init()
  rte_eal_pci_probe
  foreach(dev, pci_device_list) probe_all
```

**PMD code**

```
static struct eth_driver rte_ixgbe_pmd = {
  .pci_drv = {
    .id_table = pci_id_ixgbe_map,
    .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC |
    RTE_PCI_DRV_DETACHABLE,
    .probe = rte_eth_dev_pci_probe, !GENERIC PCI
    .remove = rte_eth_dev_pci_remove,
  },
  .eth_dev_init = eth_ixgbe_dev_init,
  .eth_dev_uninit = eth_ixgbe_dev_uninit,
  .dev_private_size = sizeof(struct ixgbe_adapter),
};
RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd.pci_drv);

static struct rte_vdev_driver pmd_null_drv = {
  .probe = rte_pmd_null_probe,
  .remove = rte_pmd_null_remove,
};
RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```
struct rte_device {
    TAILQ_ENTRY(rte_device) next; /**< Next device */
    const struct rte_driver *driver; /**< Associated driver */
    int numa_node; /**< NUMA node connection */
    struct rte_devargs *devargs; /**< Device user arguments */
};

struct rte_driver {
    TAILQ_ENTRY(rte_driver) next; /**< Next in list. */
    const char *name; /**< Driver name. */
    const char *alias; /**< Driver alias. */
};

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 */
    struct rte_intr_handle intr_handle; /**< Interrupt handle. */
    struct rte_pci_driver *driver; /**< Associated driver */
    uint16_t max_vfs; /**< sriov enable if not zero */
    enum rte_kernel_driver kdrv; /**< Kernel driver passthrough */
};

struct rte_pci_driver {
    TAILQ_ENTRY(rte_pci_driver) next; /**< Next in list. */
    struct rte_pci_dev pci_dev; /**< Device */
    struct rte_driver *driver; /**< Associated driver */
    struct rte_pci_probe_t *probe; /**< Device probe function. */
    struct rte_pci_remove_t *remove; /**< Device remove function. */
    const struct rte_pci_id *id_table; /**< ID table, NULL terminated. */
    uint32_t drv_flags; /**< Flags controlling handling of device. */
};

struct rte_vdev_driver {
    TAILQ_ENTRY(rte_vdev_driver) next; /**< Next in list. */
    struct rte_vdev_driver *vdev; /**< Associated driver for this device */
    struct rte_device *device; /**< Backing device */
    rte_vdev_probe_t *probe; /**< Virtual device probe function. */
    rte_vdev_remove_t *remove; /**< Virtual device remove function. */
};

struct rte_eth_dev {
    const struct eth_driver *driver; /**< Driver for this device */
    struct rte_device *device; /**< Backing device */
    uint8_t attached; /**< Flag indicating the port is attached */
};

struct eth_driver {
    struct rte_pci_driver pci_drv; /**< The PMD is also a PCI driver. */
    struct rte_driver *driver; /**< Driver for this device */
    struct rte_pci_probe_t *probe; /**< Device probe function. */
    struct rte_pci_remove_t *remove; /**< Device remove function. */
    unsigned int dev_private_size; /**< Size of device private data. */
};

struct rte_bus {
    TAILQ_ENTRY(rte_bus) next; /**< Next bus object in linked list */
    const char *name; /**< Name of the bus */
    rte_bus_scan_t scan; /**< Scan for devices attached to bus */
    rte_bus_probe_t probe; /**< Probe devices on bus */
};
eal init flow

rte_eal_init
  eal_parse_args
  eal_parse_common_option
  rte_eal_devargs_add
  insert(devargs_list, devargs)
rte_eal_pci_init
  insert(pci_device_list, dev)
rte_bus_scan
  foreach(bus, rte_bus_list) bus->scan() !NO BUS REGISTERED
rte_bus_probe
  foreach(bus, rte_bus_list) bus->probe()
rte_eal_pci_probe
  foreach(dev, pci_device_list) probe_all
rte_eal_devinit
  foreach(devargs, devargs_list) rte_eal_vdev_init()

Bus functions are added into eal_init, no bus registered yet

static struct eth_driver rte_ixgbe_pmd = {
  .pci_drv = {
    .id_table = pci_id_ixgbe_map,
    .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
    .probe = rte_eth_dev_pci_probe,
    .remove = rte_eth_dev_pci_remove,
  },
  .eth_dev_init = eth_ixgbe_dev_init,
  .eth_dev_uninit = eth_ixgbe_dev_uninit,
  .dev_private_size = sizeof(struct ixgbe_adapter),
};
RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd.pci_drv);

static struct rte_vdev_driver pmd_null_drv = {
  .probe = rte_pmd_null_probe,
  .remove = rte_pmd_null_remove,
};
RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
Related data structures

- PCI bus: Inherited from `rte_bus`
- vdev device struct created
- eth driver removed

---

For more information on related data structures, please refer to the original document.
```c
struct rte_pci_bus rte_pci_bus = {
    .bus = {
        .scan = rte_pci_scan,
        .probe = rte_pci_probe,
    },
    .device_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.device_list),
    .driver_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.driver_list),
};
RTE_REGISTER_BUS(PCI_BUS_NAME, rte_pci_bus);

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);

static struct rte_bus rte_vdev_bus = {
    .scan = vdev_scan,
    .probe = vdev_probe,
};
RTE_INIT(rte_vdev_bus_register);
static void rte_vdev_bus_register(void) {
    static int registered;
    if (registered)
        return;
    registered = 1;
    rte_vdev_bus.name = RTE_STR(virtual);
    rte_bus_register(&rte_vdev_bus);
}
```
eal init flow

```
rte_eal_init
  eal_parse_args
  eal_parse_common_option
  rte_eal_devargs_add
  insert(devargs_list, devargs)
  foreach(bus, rte_bus_list) bus->scan()
  RTE_BUS_SCAN
  RTE_BUS_PROBE
  foreach(bus, rte_bus_list) bus->probe()
  if (vbus) vbus->probe()
```

PMD code

```
eth_ixgbe_pci_probe
  rte_eth_dev_pci_generic_probe(pci_dev, eth_ixgbe_dev_init)

eth_ixgbe_pci_remove
  rte_eth_dev_pci_generic_remove(pci_dev, eth_ixgbe_dev_uninit)

static struct rte_pci_driver rte_ixgbe_pmd = {
  .id_table = pci_id_ixgbe_map,
  .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
  .probe = eth_ixgbe_pci_probe,
  .remove = eth_ixgbe_pci_remove,
};

RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd);

static struct rte_vdev_driver pmd_null_drv = {
  .probe = rte_pmd_null_probe,
  .remove = rte_pmd_null_remove,
};

RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```
related data structures

```c
struct rte_device {
    TAILQ_ENTRY(rte_device) next; /**< Next device */
    const char *name; /**< Device name */
    const struct rte_driver *driver; /**< Associated driver */
    int numa_node; /**< NUMA node connection */
    struct rte_devargs *devargs; /**< Device user arguments */
};
```

```c
struct rte_driver {
    TAILQ_ENTRY(rte_driver) next; /**< Next in list. */
    const char *name; /**< Driver name. */
    const char *alias; /**< Driver alias. */
};
```

```c
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; /**< 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) */
};
```

```c
struct rte_pci_driver {
    TAILQ_ENTRY(rte_pci_driver) next; /**< Next in list. */
    struct rte_driver driver; /**< Inherit core driver. */
    struct rte_pci_bus *bus; /**< PCI bus */
    struct rte_pci_driver_list driver_list; /**< List of PCI drivers */
    struct rte_pci_device_list device_list; /**< List of PCI devices */
    struct rte_pci_id *id_table; /**< ID table, NULL terminated. */
    uint32_t drv_flags /**< Flags controlling handling of device */;
};
```

New bus functions for hotplug support

Updated devargs, removed bus-specific name or address
Two new APIs:
- `rte_eal_hotplug_add`
- `rte_eal_hotplug_remove`


**eal init flow**

- `rte_eal_init`
  - `eal_parse_args`
    - `eal_parse_common_option`
      - `rte_eal_devargs_add`
        - `insert(devargs_list, devargs)`
  - `rte_bus_scan`
    - `foreach(bus, rte_bus_list) bus->scan()`
  - `rte_bus_probe`
    - `foreach(bus, rte_bus_list) bus->probe()`
    - `if (vbus)`
      - `vbus->probe()`

**PMD code**

- `eth_ixgbe_pci_probe`
  - `rte_eth_dev_pci_generic_probe pci_dev, eth_ixgbe_dev_init())`

- `eth_ixgbe_pci_remove`
  - `rte_eth_dev_pci_generic_remove pci_dev, eth_ixgbe_dev_uninit)()`

- static `struct rte_pci_driver` `rte_ixgbe_pmd` = {
  - `.id_table = pci_id_ixgbe_map`,
  - `.drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC`,
  - `.probe = eth_ixgbe_pci_probe`,
  - `.remove = eth_ixgbe_pci_remove`,
}

- `RTE_PMD_REGISTER_PCI net_ixgbe, rte_ixgbe_pmd);`

- static `struct rte_vdev_driver` `pmd_null_drv` = {
  - `.probe = rte_pmd_null_probe`,
  - `.remove = rte_pmd_null_remove`,
}

- `RTE_PMD_REGISTER_VDEV net_null, pmd_null_drv);`
related data structures

NO change (yet)
bus code

```c
struct rte_pci_bus rte_pci_bus = {
    .bus = {
        .scan = rte_pci_scan,
        .probe = rte_pci_probe,
        .find_device = pci_find_device,
        .plug = pci_plug,
        .unplug = pci_unplug,
        .parse = pci_parse,
    },
    .device_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.device_list),
    .driver_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.driver_list),
};
RTE_REGISTER_BUS(pci, rte_pci_bus.bus);

struct rte_fslmc_bus rte_fslmc_bus = {
    .bus = {
        .scan = rte_fslmc_scan,
        .probe = rte_fslmc_probe,
        .find_device = rte_fslmc_find_device,
    },
    .device_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.device_list),
    .driver_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.driver_list),
    .device_count = 0,
};
RTE_REGISTER_BUS(fslmc, rte_fslmc_bus.bus);

static struct rte_bus rte_vdev_bus = {
    .scan = vdev_scan,
    .probe = vdev_probe,
    .find_device = vdev_find_device,
    .plug = vdev_plug,
    .unplug = vdev_unplug,
    .parse = vdev_parse,
};
RTE_REGISTER_BUS(vdev, rte_vdev_bus);

struct rte_dpaa_bus rte_dpaa_bus = {
    .bus = {
        .scan = rte_dpaa_bus_scan,
        .probe = rte_dpaa_bus_probe,
        .find_device = rte_dpaa_find_device,
    },
    .device_list = TAILQ_HEAD_INITIALIZER(rte_dpaa_bus.device_list),
    .driver_list = TAILQ_HEAD_INITIALIZER(rte_dpaa_bus.driver_list),
    .device_count = 0,
};
RTE_REGISTER_BUS(FSL_DPAA_BUS_NAME, rte_dpaa_bus.bus);
```

New bus driver
eal init flow

rte_eal_init
  eal_parse_args
    eal_parse_common_option
      rte_eal_devargs_add
        insert(devargs_list, devargs)
  rte_bus_scan
    foreach(bus, rte_bus_list) bus->scan()
  rte_bus_probe
    foreach(bus, rte_bus_list) bus->probe()
      if (vbus)
        vbus->probe()

eth_ixgbe_pci_probe
  rte_eth_dev_pci_generic_probe(pci_dev, eth_ixgbe_dev_init)

eth_ixgbe_pci_remove
  rte_eth_dev_pci_generic_remove(pci_dev, eth_ixgbe_dev_uninit)

static struct rte_pci_driver rte_ixgbe_pmd = {
  .id_table = pci_id_ixgbe_map,
  .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
  .probe = eth_ixgbe_pci_probe,
  .remove = eth_ixgbe_pci_remove,
};

RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd);

static struct rte_vdev_driver pmd_null_drv = {
  .probe = rte_pmd_null_probe,
  .remove = rte_pmd_null_remove,
};

RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
- Move vdev and PCI into drivers/bus/ folder (patches on patchwork).
  - Looking for volunteers! For drivers/bus/* maintainership.

- Devargs rework separate it from specific bus.
  - whitelist / blacklist must be more generic (not only PCI).

- drivers/bus/net?
- Hyper-V VMBUS?
- Add/Remove notifications for hotplug.
- More explicit and extensible devargs.
- Remove rte_bus reference from devargs.
- Bus documentation?
Thanks

- Shreyansh Jain
- Jan Blunck
- Gaetan Rivet
- And rest that I missed.
Questions?

Ferruh Yigit
ferruh.yigit@intel.com